5-2017

A Hybrid Quantum Random Number Generation Methodology to Insure Secure Key

Karthik Paidi
St. Cloud State University, kpaidi@stcloudstate.edu

A Hybrid Quantum Random Number Generation Methodology to Insure Secure Key

by

Karthik Paidi

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

May, 2017

Starred Paper Committee:
Dr. Dennis Guster, Chairperson
Dr. Renat Sultanov
Dr. Balasubramanian Kasi

Abstract

In the world of computation and digital communications the digital world is currently lacking in ‘security.' Yes, security is a feature that can never be attained one hundred percent. However, to ensure secure data we can use huge numbers and large cryptographic keys in combination with a statistical algorithm so that deceiving or decryption of information would become very difficult. The question then becomes what if someone reaches a level in computational speed like none other with the support of advanced chip technology and cracks all the available mathematical algorithms built in combination with the available cryptographic keys? Then the world of digital computation, which makes us feel secure, becomes at risk. Recent research and achievements in advanced technology, especially in Quantum Computation and Encryption, are ringing danger bells towards conventional computational security methodologies. In this paper, I will discuss current security trends, advancements in quantum computation and traditional computation security methods that feel insecure and discuss a new methodology that uses the spin rotation of photons to add the power of quantum mechanics to classical encryption algorithms to insure a balanced key generation.

Acknowledgement

This research paper about designing and implementing Quantum security, a hybrid quantum algorithm was undertaken using the resources provided by the Business Computing Research Laboratory of St. Cloud State University.

Table of Contents

List of Figures

Page

Chapter 1: Introduction

Introduction

These days it is unimaginable to live without computational devices whether they are large scale or small scale. These devices help humans make life easier, with fewer efforts than usual, and shows us that they have become an integral part of our lives. The usage of these devices includes personal to the professional life of each person, which involves huge amounts of data transmission between individuals within the range of low to high level of confidentiality. Cryptography derives different ways of security mechanisms through which maximum amount of secure communication can be assured.

There are several mechanisms followed to achieve this security by encrypting the data and transfer over a secure medium which requires different sizes of keys to encrypt data and secure media to transfer it. As the size of cryptographic keys increase the chances of deceiving through the encryption of data within the time frame decreases. Let's not infer that extensive size of keys will keep our data secure as the computational capacity and speed of machines increases, there is a fair chance of breaking the key and allowing for the decryption of data. Let us assume then that there is a key of size of 16 bits, which is used for encryption of data and to break the key 65536 combinations are required. At one point a hacker will get the key while trying these combinations, it would be easy to break into your information with the key by using the functionality of the algorithm. By looking at the above example of how the AES (Advanced Encryption Standard) algorithm with 16 bits can be broken, it could be inferred that advancement in the computational speed in the future will lead to breaking all the keys in a minimal amount of time.

Moore’s law “is the observation that, over the history of computing hardware, the number of transistors in a dense integrated circuit has doubled approximately every two years” [1][2][3].

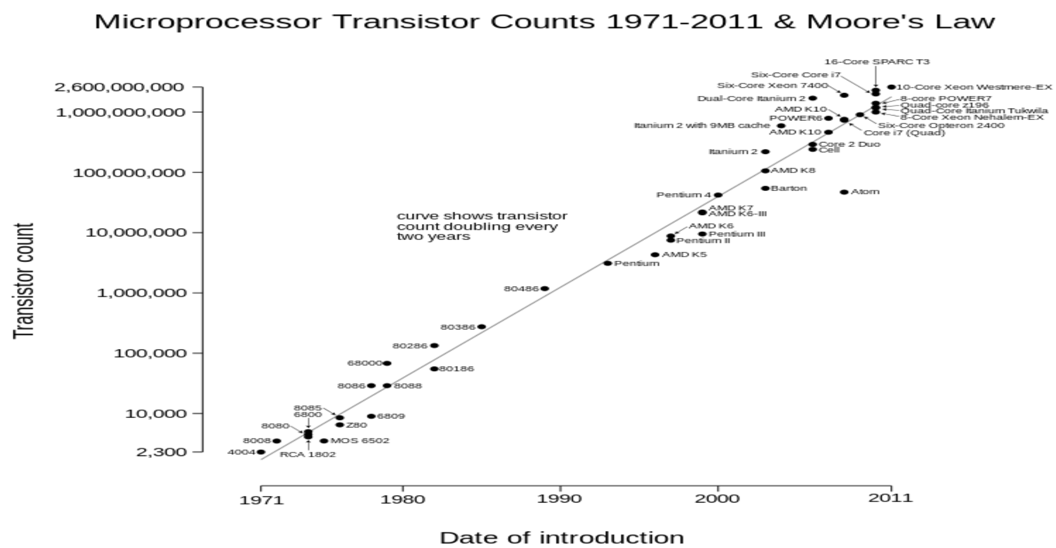


Figure 1. Moore’s Law Graph

If Moore’s law is kept in mind with greater observation it could be inferred that technological advancement in achieving the high computational speed over the years, there will be a time where manufacturing of extensive high-speed processors [2][3] which in combination can give birth to new systems that can process millions of combinations in an optimal amount of time. All the above assertions imply how much danger the world of cryptography could be potentially in the near future.

Apart from classical computation, there were new theories proposed and partially implemented in the 20th century to advance computational speed as well as security (i.e. cryptography) which, uses physics concepts, such as Quantum Mechanics. This new era of quantum computation/encryption made its significant achievements at the beginning of the 21st century, which gave birth to the new advanced computers called Quantum Computers that provide exceptional speed by using Qbits (Quantum bits). Though these computers have issues in a full-scale implementation like distance, temperature, and components however its efficiency made a significant impact in traditional computation. There are many ongoing pieces of research conducted by utilizing the ability of quantum computation with conventional methodologies through which it has informed our traditional methods of computation are in danger.

Though quantum computers are still experimental and in 2008 the largest, so far, is a 16 Qbit system built by D-Wave in Canada [4] (Double check on requirements for long quotes with the citation method you used)"Called Orion, it is a superconducting adiabatic quantum computer. The main computing engine is held in a big red tank, supercooled to a frosty 4mK (0.004 degrees Celsius above absolute zero, colder than interstellar space!) with liquid helium. The core computational unit is a single chip, with 16 Qubits arranged in a four by four grid. Each Qbits is coupled directly to its immediate neighbors (North, South, East, and West) and those on the diagonal, which provides considerably less efficiency than the theoretical maximum of every Qubit entangled to every other Qubit." [5]. Imagine if a quantum computer matches the speed of a supercomputer with the fewer number of cores then how easy to break the keys by trying the various combinations. There are so many advantages over the disadvantages as QKD [7] (Quantum Key Distribution) is a proven methodology to do a secure transmission of the key over the communication channel. Where there is no chance of eavesdroping and the random number generator by IDQ [6], which made the greatest impact on other random number generation algorithms that are in use.

Further, this paper will discuss what quantum computation/encryption is, why we need true random number generators, how a quantum random number generator works, and what is the base concepts used to prove the generation of conventional quantum random number generator algorithm.

Problem Statement

Case 1

Till now this paper discussed what a quantum computer, is and how fast it can perform computations. Just imagine if the computer built using quantum concepts and was able to process data at a speed of 100 high-speed conventional computers then it would be easy for people holding these computers in their possession to do things in minutes. If the possessor is a bad guy (Hacker) then they can do 100 computers worth of work with this single computer in even less time, if they wanted to get into another computer unethically to steal some information, then the entire process will become very easy for them.

Case 2

After taking multiple actions towards putting in protections against hackers and viruses still, they can break in and steal the information and after processing (Decrypting) the stolen information they would be able to do illegal work using other peoples' identities. Even after having huge cryptographic key's to encrypt data, still hackers can break in. So, it is necessary to find new ways to protect our data from falling into their hands.

Nature and Significance of the Problem

Above the paper has stated two problems both of them might sound different but the theme of the problems is a similar breaking in and stealing the information and processing it to use for another's benefit. In these cases one thing can be inferred is that it is necessary to use a new technique to handle the quantum computers, start new ways to hide the information so that it should be more difficult to process the information for a hacker who comes into possession of it.

Objective of the Research

The objective of this research is to state the problems now faced by the security world in traditional computation, provide a theoretical solution to this issue such that a hacker will never know what the base methodology followed to encrypt the information is. Even after getting the information of the processed (Encrypted) data, which gives us a fair chance to feel safe, even if it is stolen. Here this paper proposes a hybrid algorithm, which follows the rules of quantum mechanics and can be implemented and used in conventional computers.

Summary

In this chapter we discussed how the technology has evolved, how newly developed methodologies can benefit us with their incredible computation powers and how they are raising danger bells towards our existing conventional methods. Eventually, this paper will discuss a new method, which uses the same concepts and gets utilized in the existing systems to reduce these effects at least partially.

Chapter 2: Background and Review of Literature

Introduction

The work of Bennett and Brassard, 1984 [14] provided a practical means of deploying data transmission using quantum keys. This protocol, which has become known as BB84 in its original form used photon polarization states as the transmission logic. From a quantum perspective, any two pairs of conjugate states can be used to support the protocol. Because several optical fiber based implementations have been devised to use phase based encoding the practicality of this method has increased. Further refinements in the form of a two-step process of this basic BB84 logic have followed. These two steps, described by Bennett et al., 1992, first presented information reconciliation and privacy amplification [15]. Briefly, information reconciliation can be viewed as a form of error correction carried out during the key exchange, which is designed to ensure that both keys are identical. For more information about a sample protocol using this technique see Brassard and Salvail, 1993 [16].

The second step has been deemed privacy amplification, which is a method for almost removing any partial information that might be obtained about the key by an eavesdropper. Specifically, privacy amplification takes the actual key and modifies it to confuse a hacker. Often the resulting key is shorter, which provides a potential eavesdropper with only minimal information about the new key. This process is often accomplished by using a universal hash function. For more information concerning this process please refer to, Kaser and Lemire, 2013 [17]. It is a variant in the privacy application process, which is the main focus of this paper.

While the development of a full-scale Internet style quantum encrypted network is still some time off there have been commercial successes. Of particular note would be the work of DARPA (Quantum, 2005), Id Quantique [18] and Los Alamos National Laboratory [19]. While it is generally accepted that the quantum-based systems offer enhanced security beyond classical solutions in part because hacking attacks can be detected. The hesitation to adopt them comes from a high equipment cost and perceived lack of need. However, it is undeniable that quantum computers continue to progress and exhibit computing speeds that are significantly faster that classical computers [20]. Kirsch, 2015 [21] puts the danger that quantum computing poses to classical encryption methods such as RSA into perspective: “a quantum computer can factor a 300 digit number in the same amount of time that an ordinary computer could multiply the factor together, rendering our current encryption methods obsolete”.

Therefore, in the meantime hybrid algorithms are needed as a stop-gap measure to protect against quantum brute force attacks designed to compromise the encryption key. This is thus the main focus of this paper. In production systems there is often an option of combining a QKE unconditionally secure key exchange sub-system with traditional encryption algorithms such as 3DES or AES [22], [23] while this type of hybrid system cannot be considered unconditionally secure it still offers some security advantages over traditional purely classical strategies. Specifically, the public key authentication mechanism would have to be broken before or during the execution of the QKE protocol [24]. Work with hybrid quantum keys continues to appear in the literature and in many cases the goal is to use a quantum generator and then use mathematical functions to obscure the key further. A recent example of this approach is presented by Lai, Xue, Orgun, Xiao and Pieprzyk, Feb. 2015 [25]. They devised a protocol that applies extended unitary operations derived from four basic unitary operations and distributed fountain codes. When testing this protocol they found it to be highly efficient, secure and as planned it provides authentication of parties and detection of eavesdropping.

Because of the effectiveness of hybrid QKD protocols in preventing attacks in the quantum channel a recent work describes the value of applying it to wireless communication. Nail and Reddy, 2015 [26] devised new scheme with the combination of quantum cryptography and classical cryptography for 802.11i wireless LANs. This ground-breaking work demonstrated the value and transferability of quantum cryptography and can be viewed as a significant step forward toward securing communications in wireless networks. When tested, the hybrid quantum key distribution protocol they devised added robustness in securing wireless networks.

In sum it is clear that quantum encryption can offer distinct advantages over purely classical solutions. While the development of large numbers of large-scale quantum computers is still some time away the problem of current classical algorithms becoming obsolete cannot be ignored. Therefore, stopgap solutions such as hybrid algorithms are still important and it is hoped that the hybrid algorithm offered herein will contribute to the understanding of such concepts both operationally and educationally.

Background Related to the Problem

Most of the conventional algorithms used presently in the world are bound with some kind of key, which will be kept secret in order to make the process of decryption using the same algorithm difficult. Even though it is secret, hackers still are able to decrypt them using highly configured computer equipment. In order to save the key from these issues some new generation algorithms are needed, which use new concepts that require intensive study to know how they work.

Literature Related to the Problem and Methodology

Quantum computer basics. Until now we have not discussed the binary bits, which are the basis of computers input and output. A binary bit can be either ‘0’ or ‘1’ that means for every single bit generation there is a probability of two that is 0 or 1. But Qbits are different from regular bits in that it is a combination of the 0 or 1[8] [9] [10]. It might be amazing to know that there is another bit that exists that can perform computation operations and yes, it is true and is named the Qbit (Quantum bit). A Qbit is an overlapped bit of 0/1 i.e. a quantum computer uses ‘0’,’1’ and ‘Qbit’. A two Qbit system can perform the operation on four values so by obtaining the quantum parallelism with a proper algorithm problem of conventional computers can be solved within seconds.

Superposition. The superposition principle is the idea that a system is in all possible states at the same time until it is measured. After measurement it then falls to one of the primary states that form the superposition, thus destroying the original configuration [11][12].

Qbits. As quantum mechanics says that any system can exist in a super positioned state, a Qbit is a state of super position of more than one bit and in general it is showed as by the following.

$$\propto |0| + \beta|1|$$

Where alpha ($\propto$) and beta (β) are the complex numbers satisfying $| \propto |^2 + |\beta|^2=1$

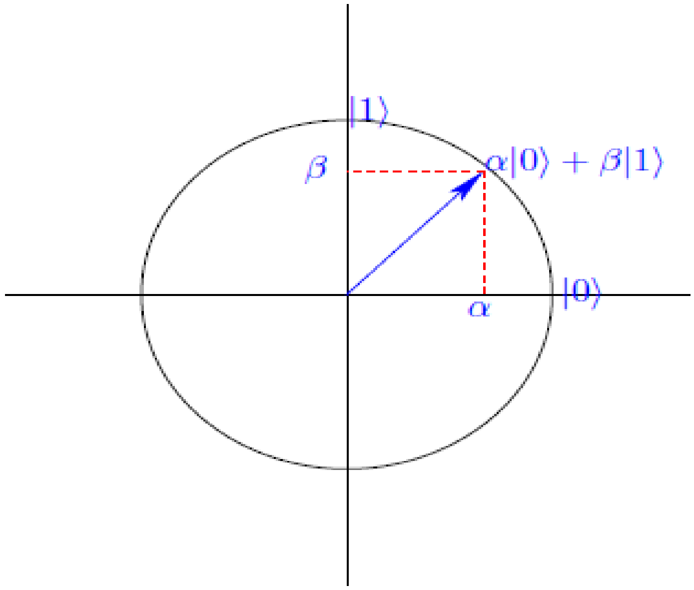


Figure 2: Vector Visualization of Qbit

Thus it seems that Qbits can hold exponentially more information than classical bits but it is not accurate. In actuality, a probabilistic superposition referred to as a probability wave, and as long as the Qbits remain undisturbed, they are thought to hold all probable values, this is known as quantum indeterminacy [13]. Though the moment a Qbit is measured the probability wave collapses into a single outcome. So, the trick of the quantum computation is the manipulation to get the desired result.

Summary

In the literature review, this paper discussed how the quantum computation idea was developed and when the actual conventional implementation started in this field. Also, the basics of what the base technology of quantum computation that has been developed, what are Qbits, and what is meant by superposition and quantum indeterminacy.

Chapter 3: Methodology

Introduction

In this chapter implementation of the conventional quantum concept based algorithm is expanded further. This paper expects readers to think of the flow of the algorithm in a deterministic way but instead it was supposed to be non-deterministic as implementation is for the conventional computers.

Design of the Study

The base for this study is a real world scenario, which is problematic from a security standpoint, and also effects privacy of information or data. It has been a known fact that every encryption algorithm works with public and private keys to ensure the encryption and decryption are taking place at the right place, with the right person. If someone knows the key used to decrypt the message then he or she can tamper with the personal, classified data. It shows us the importance of the key in encryption and decryption, thus if there is a possibility to seal or create the key in such a way even after knowing the part of the key could not help in breaking the full key then users are at an advantage.

By considering all these factors, I have decided to go with an approach, which looks like non-deterministic at each step but it is deterministic and a new approach, which will need minimal knowledge of quantum physics apart from regular math and programming analysis to understand the workflow.

Data Collection Data Analysis

Most of the time to complete this approach I followed IDQ random number generator work functionality, BB84 protocol work functionality and behavioral properties of the photon. I tried to pull the relation between each of them and worked on the working relations keeping the existing methodologies in mind to come up with this solution.

Tools and Techniques

I did not use any tools to complete this study, but I used existing random number generators like the IDQ random number generator, API based random number generation functions of object oriented programming to observe the quality and quantity of random data that is getting generated.

Hardware and Software Environment

PROCESSOR	Intel/AMD
RAM	1GB
DISK SPACE	100GB
OPERATING SYSTEM	windows professional/MAC/Linux
PROGRAMMING LANGUAGE	JAVA
VERSION	JDK1.6 & above
IDE	ECLIPSE

Chapter 4: Implementation

Flowchart

The following flow diagram of the algorithm will explain step-by-step execution of the process of key generation with respect to the input.

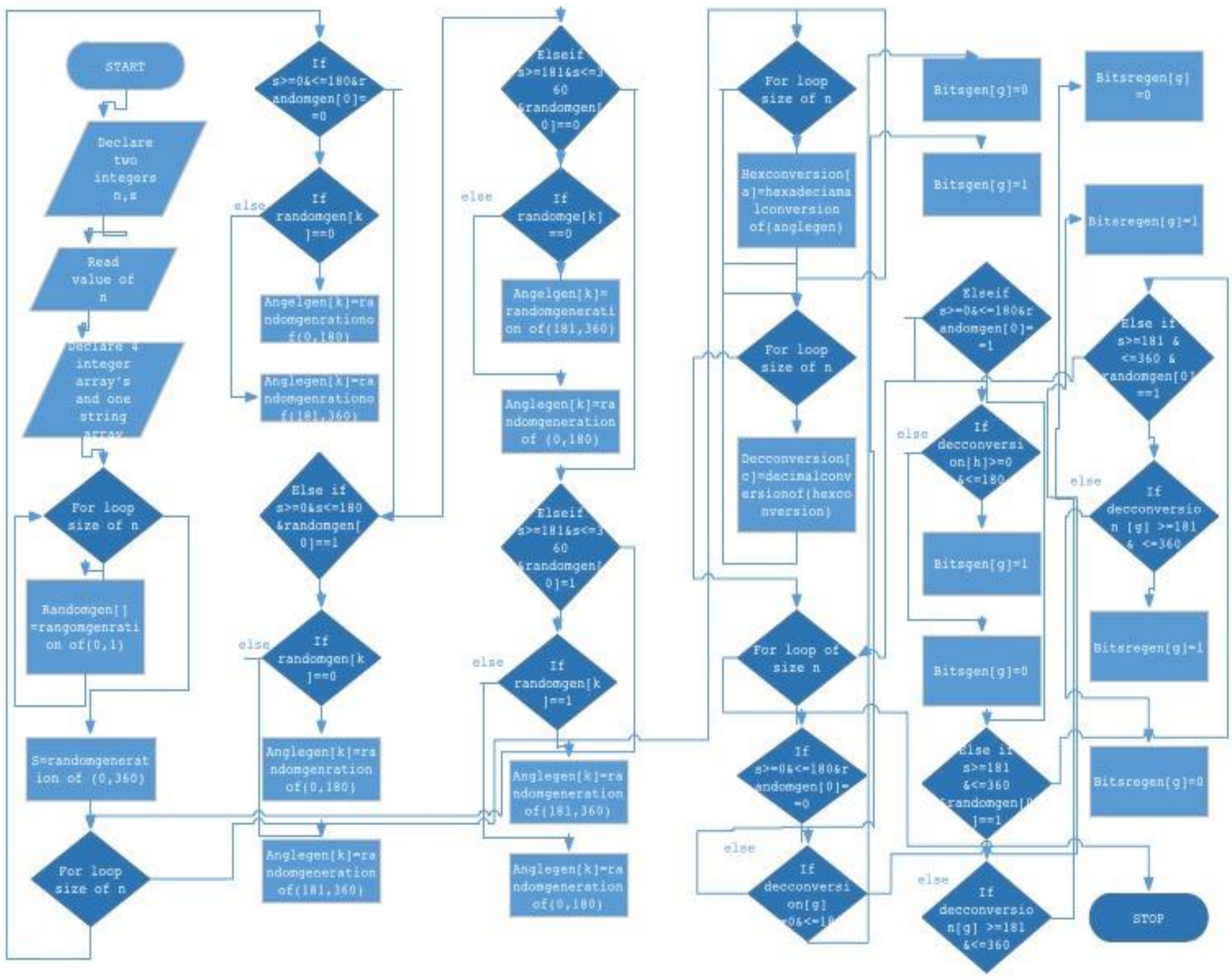


Figure 3. Explanation of the Flow of the Algorithm

Algorithm

Step1: START

Step2: Declare two integer type variables: n, s

Step3: Declare 4 integer arrays, i.e. randomgen, anglegen, decconversion, bitsregen and one string type array hexconversion

Step4: Using the scanner method scan the number of bits the user wants to generate i.e. n

Step5: Write a For loop that runs for n

For loop starts

Step6: To generate the random bits call a method that can generate random bits of a given size. (here it is a recursive function)

Step7: Define a method to get random binary numbers within a range with min, max as integer arguments and inside use Random () predefined and return the bits generated in the range of 0,1 (here min =0 and max =1)

Step8: Store the randomly generated bits to an array randomgen

For loop closes when the bits are generated for the given size

Step9: Run a spin which selects any one number randomly from 0 to 360 and store the spin in to (S) integer and to this random number define a method spinforfirstangle with min=0, max=360 as arguments

Step10: To generate respected angles for the generated random bits declare for loop for size n.

For loop starts

Step11: Check for the condition if spin s>=0 and s<=180 and generated first random bit randomgen [0] == 0 then check for another condition if randomgen of kth bit == 0 then store the angles into array anglegen and those angles are from 0-180 to generate random angles form 0-180 write the same kind of function in Step7 but the min and max are 0,180 i.e all the 0's of the generated bits are now in range of 0-180

Step12: Else case is for the 1's where those bits belong to 181-360 to generated the random angles from 181-360 use method angbetoneeightoneandthreesixyt and the arguments are 181 and 360, i.e. all the 1's belongs to 181-360 in this case

Step13: Else if check for the condition spin generated is in s>=0 and s<=180 and first randomly generated bit randomgen [0] == 1 then check for condition if randomgen [kth bit] == 1 then store the angles generated from 0-180 and stored into anglegen and generate those angles using method in Step11 i.e. all the 1's generated will be in the range of 0-180

Step14: Else case is for the 0's where those bits belongs to 181-360 to generated the random angles from 181-360 use method angbetoneeightoneandthreesixyt and the arguments are 181 and 360, i.e. all the 0's belongs to 181-360 in this case

Step15: Else if check for the condition if spin s>=181 and s<=360 and generated first random bit randomgen [0] == 0 then check for another condition if randomgen of kth bit == 0 then store the angles into array anglegen and those angles are from 181-360 to generate random angles form 181-360 write the same kind of function in Step7 but the min and max are 181, 360 i.e all the 0's of the generated bits are now in range of 181-360

Step16: Else case is for the 1's where those bits belongs to 0-180 to generate the random angles from 0-180 use method anglebetzandoneeight and the arguments are 0 and 180, i.e. all the 1's belongs to 0-180 in this case

For loop is closed

Step17: Run a for loop of size n to convert the generated angles stored in anglegen array to hexadecimal

Step18: Write the predefined or new user defined function to convert the decimal code of angles to hexadecimal code and store them to hexconverion array

Step19: Now we have to reverse the procedure to get the old bits for which, we converted the hexadecimal code to decimal and store them in to an integer array

Step20: After getting the decimal values declare a for loop of size n to convert those angles back to normal randomly generated bits

For loop starts

Step21: Check for the condition if the first spin s>=0 and s<=180 and first bit randomgen [0] == 0 then check for condition if decconversio [gth bit] >=0 and<=180 then all the angles in the range of 0-180 are 0's else the bits that are going to be regenerated will be 1's for all the other angles i.e (181-360)

Step22: Else if check for the condition if the first spin is s>=0 and<=180 and first bit randomgen [0] ==1 then check for the condition if decconversion [gth bit] >=0 and <=180 then all the angles in range of 0-180 will become as 1's else the rest of the angles will become 0's i.e (181-360)

Step23: Else if the first spin generated s>=181 and <=360 and the first bit generated randomgen [0] ==0 then check for the condition if decconversion [gth bit] >=181 and <=360 then all the angles in the range of 181-360 converted to 0's else other angles are converted to 1's i.e 0-180

Step24: Else if the first spin generated s>=181 and <=360 and the first bit generated randomgen [0] ==1 then check for the condition if decconversion [gth bit] >=181 and

<=360 then all the angles in the range of 181-360 converted to 1’s else other angles are converted to 0’s i.e 0-180

For loop closed

Pseudo code

Step1:

Declare two integers n,s

Declare four integer arrays Randomgen [], anglegen[],decconversion[],bitsregen[]

Declare a string array hexconversion []

Step2:

Scan for n i.e how many integers needed

Step3:

a) Run a for loop of size n

b) randomgen[]= (int) (getRandomNumberInRange(0,1));

c) End for loop

Step4:

Assign a spin to integer s = spinforfirstangle [of range (0,360)]

Step5:

a) Run a for loop of size n

b) Check for condition if(s>=0 && s<=180 && randomgen [0] == 0

a. Then

b. Check for the condition if (randomgen [] == 0)

i. Then

ii. Assign anglegen [] = (int) (angbetzandoneeight [of range (0,180))]

1. Else

2. anglegen [ ]= (int)(angbetoneeightoneandthreesixty[of range (181,360))]

c) End if

d) End if

e) Else if check for (s>=0 && s<=180 && randomgen [0] == 1) Then

f) Check for condition if (randomgen [ ] == 1)

a. Then

b. anglegen[ ]=(int) (angbetzandoneeight[of range(0,180))]

i. else

ii. anglegen[ ]= (int) (angbetoneeightoneandthreesixty[of range (181,360))]

c. End if

g) End else if

h) else if check for(s>=181 && s<=360 && randomgen[0] ==0 Then

check for condition if(randomgen[ ] == 0)

i. Then

b. anglegen[]=(int)(angbetoneeightoneandthreesixty[of range (181,360))]

i. else

ii. anglegen[k]= (int) (angbetzandoneeight[of range(0,180))]

c. End if

i) End else if

j) else if check for (s>=181 && s<=360 && randomgen[0] == 1 Then

a. Check for if(randomgen[ ] == 1)

b. Then

c. anglegen[ ]= (int) (angbetoneeightoneandthreesixty[of range (181,360))]

i. else

ii. anglegen[ ]= (int) (angbetzandoneeight[of range(0,180))]

d. End if

k) End else if

l) End of for loop

Step6:

a) Run a for loop of size n

a. hexconversion[ ] = Integer.toHexString(anglegen[ ])

b) End of loop

Step7:

a) Run a for loop of size n

a. decconversion[] = Integer.parseInt(hexconversion[ ],16)

b) End of loop

Step8:

a) Run a for loop of size n

a. Check for if(s>=0 && s<=180 && randomgen[0] == 0 Then

i. Check for if(decconversion[ ]>= 0 && decconversion[ ] <=180)

a. Then

bitsregen[]= 0

ii. else

iii. bitsregen[ ]= 1

b. End if

b) End if

c) Check for else if(s>=0 && s<=180 && randomgen[0] == 1) Then

Check for if(decconversion[ ]>= 0 && decconversion[ ] <=180)

i. Then

ii. bitsregen[ ]= 1

iii. else

iv. bitsregen[ ]= 0

b. End if

d) End else if

e) else if check for (s>=181 && s<=360 && randomgen[0] ==0) Then

f) Check for if(decconversion[ ]>= 181&&decconversion[ ] <=360)

1. Then

a. bitsregen[ ]= 0

ii. else

1. bitsregen[ ]= 1

b. End if

g) End else if

h) else if check for (s>=181 && s<=360 && randomgen[0] == 1 Then

a. Check for if(decconversion[g]>= 181 && decconversion[g] <=360)

i. Then

ii. bitsregen[ ]= 1

b. else

i. bitsregen[ ]= 0

c. End if

i) End else if

j) End of for loop

Step9:

a) write a function getRandomNumberInRange(int min, int max)

i. create a onstructor r for Random

ii. return r.nextInt((max - min) + 1) + min

b) End of function

Step10:

a) write a function spinforfirstangle(int min, int max)

i. create a onstructor r for Random

ii. return r.nextInt((max - min) + 1) + min

b) End of function

Step11:

a) write a function angbetzandoneeight (int min, int max)

i. create a onstructor r for Random

ii. return r.nextInt((max - min) + 1) + min

b) End of function

Step12:

a) write a function angbetoneeightoneandthreesixty (int min, int max)

i. create a onstructor r for Random

ii. return r.nextInt((max - min) + 1) + min

b) End of function

Code

```
import java.util.Random;
import java.util.Scanner;
public class RNG
{
public static void main(String[] args)
{
int n;//number of bits you need
int s;//first angle
Scanner in=new Scanner(System.in);// scanner to scan n
System.out.print("enter how many bits you want to generate\n");
n=in.nextInt();//scans n

int[] randomgen =new int[n];//array to store generated random bits
int[] anglegen =new int[n]; // array to store angles with respect to bits
        String[] hexconversion =new String[n];// string array to store the hexadecimal value of the
generated angles
int [] decconversion =new int[n]; // integer array to store the decimal converted hex values of angels
int [] bitsregen =new int[n]; // integer array to store the decapsulated bits

//Encapsulation starts
                        for(int i=0;i<n;i++)
                                randomgen[i]= (int) (getRandomNumberInRange(0,1)); //storing bits to
randomgen array

                                        for(int j=0;j<n;j++)
                                                System.out.print(" "+randomgen[j]);//printing
randomgen bits from array

                                        //spin angle for first bit
                                        s=spinforfirstangle(0,360);
                                                System.out.print("\nthe angle associated for the first bit
is:"+" "+s);
```

```
System.out.print("\n");
//decision to make which set of angles will become 1 or 0
for(int k=0;k<n;k++)// loop to run anglegen and randomgen arrays
{
if(s>=0 && s<=180 && randomgen[0] == 0)// if the first spin is above 0 and below or equal to 180 and first bit 0
{
//System.out.println("\n 0 and is in below 180 :"+s);// just to test loop is working properly or not
if(randomgen[k] == 0)
{
anglegen[k]= (int) (angbetzandoneeight(0,180)); // saving the angles to array with respect to the first spin
}
else
{
anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin
}
}
else if( s>=0 && s<=180 && randomgen[0] == 1) // else if first spin is above 0 and below or equal to 180 and first bit 1
{
//System.out.println("\n1 and is in below 180 :"+s);// just to test loop is working properly or not
if(randomgen[k] == 1)
{
anglegen[k]=(int) (angbetzandoneeight(0,180)); // saving the angles to array with respect to the first spin
}
else
{
anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin
}
}
else if(s>=181 && s<=360 && randomgen[0] ==0)// else if first spin is above 181 and below or equal to 360 and first bit 0
{
//System.out.println("\n0 and is in above 181 :"+s);// just to test loop is working properly or not
if(randomgen[k] == 0)
```

```
{
anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin
}
else
{
anglegen[k]= (int) (angbetzandoneeight(0,180)); // saving the angles to array with respect to the first spin
}
}
else if(s>=181 && s<=360 && randomgen[0] == 1)// else if first spin is above 181 and below or equal to 360 and first bit 1
{
//System.out.println("\n1 and is in above 181 :"+s);// just to test loop is working properly or not
if(randomgen[k] == 1)
{
anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin
}
else
{
anglegen[k]= (int) (angbetzandoneeight(0,180));// saving the angles to array with respect to the first spin
}
}

}

//printing of the angles generated in association with the bits are
        System.out.println("the angles generated with respect to bits are :");
for(int f=0;f<n;f++)
        System.out.println(anglegen[f]);//printing the angles

//iterative loop to convert angles to hex code
for(int a=0;a<n;a++)
{
        hexconversion[a] = Integer.toHexString(anglegen[a]);//conversion and storage of the converted angles to string array
}

System.out.println("Hex code of the converted angles :");
//printing of the converted hexcode of the angles
for(int b=0;b<n;b++)
        System.out.println(" "+hexconversion[b]);//print statement to converted Hexangles
```

```
//for encryption use existing algorithms which can raise the magnitude of trustworthy security of your bit key
        //decapsulation starts
    //revesing back from hex to decimal values(angles genrated)
    for(int c=0;c<n;c++)
        decconversion[c] = Integer.parseInt(hexconversion[c],16);//conversion of string to integer

    System.out.println("Decimal values of the hex code :");

    //printing back converted decimal
    for(int d=0;d<n;d++)
        System.out.println(" "+decconversion[d]);//printing of the decimal values of the hex

//converting back to bits generated
    for(int g=0;g<n;g++)// loop to run anglegen and randomgen arrays
    {
        if(s>=0 && s<=180 && randomgen[0] == 0)//spin in 0-180 and first bit is 0
        {
            if(decconversion[g]>= 0 && decconversion[g] <=180)// angle generated is in range of 0-180
            {
                bitsregen[g]= 0 ;// depending on first bit 0
            }
            else
            {
                bitsregen[g]= 1;// if first bit its not zero they fall under 1
            }
        }
            else if( s>=0 && s<=180 && randomgen[0] == 1) // else if spin in 0-180 and first bit is 1
        {
            if(decconversion[g]>= 0 && decconversion[g] <=180)// angle generated is in range of 0-180
            {
                bitsregen[g]= 1 ;// depending on first bit 1
            }
            else
```

```
{
bitsregen[g]= 0;// if first bit its not one they fall under 0
}
}
else if(s>=181
&& s<=360 && randomgen[0] ==0)// else if spin in 181-360 and first bit is 0
{
if(decconversion[g]>= 181 && decconversion[g] <=360)// angle generated is in range of 0-180
{
bitsregen[g]= 0 ;// depending on first bit 0
}
else
{
bitsregen[g]= 1;// if first bit its not zero they fall under 1
}
}
else if(s>=181
&& s<=360 && randomgen[0] == 1)// else if spin in 181-360 and first bit is 0
{
if(decconversion[g]>= 181 && decconversion[g] <=360) // angle generated is in range of 0-180
{
bitsregen[g]= 1 ;// depending on first bit 1
}
else
{
bitsregen[g]= 0;// if first bit its not one they fall under 0
}
}
}
System.out.println("Decapsulated bits from angels are :");
//printing the decapsulated random gen bits
for(int e=0;e<n;e++)
System.out.println(" " + bitsregen[e]);//prints the exact
bits used for key generated
}// main class close

// function to generate random bits
private static int getRandomNumberInRange(int min, int max) {
```

```
//exception case for number generation logic
if (min >= max) {
        throw new IllegalArgumentException("max must be greater than min");
}

Random r = new Random();//random generating internal function

return r.nextInt((max - min) + 1) + min;// returning of generated numbers
}

// function to generate random number i.e first angle
        private static int spinforfirstangle(int min, int max) {

                //exception case for number generation logic
                if (min >= max) {
                        throw new IllegalArgumentException("max must be greater than min");
                }

                Random r = new Random();//random generating internal function

                return r.nextInt((max - min) + 1) + min;// returning of generated first associated
angle
        }

                //function to generate angles between 0 and 180
                private static int angbetzandoneeight(int min, int max) {

                        //exception case for number generation logic
                        if (min >= max) {
                                throw new IllegalArgumentException("max
must be greater than min");
                        }

                        Random r = new Random();//random generating internal
function

                        return r.nextInt((max - min) + 1) + min;// returning of
generated first associated angle
                }

                        //function to generate angle between 181 and 360
                                private static int angbetoneeightoneandthreesixty(int
min, int max) {

                                        //exception case for number generation
logic
                                        if (min >= max) {
                                                throw new
IllegalArgumentException("max must be greater than min");
                                                }
```

```
Random r = new
Random();//random generating internal function

return
r.nextInt((max - min) + 1) + min;// returning of generated first associated angle
}
}//class close
```

Testing and Execution

To test the workflow of the algorithm Dr.Dennis Guster prepared random 168 bits, which are generated randomly using a pseudo random bit generator and used as the input for the algorithm. The output of the algorithm we got is satisfying all of the conditions mentioned in the flow diagram and generating random angles by basing the spin bit.

The below 168 bits are given as the sample test bits:

1,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,1,0,1,0,0

0,0,0,1,1,0,0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0

1,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,1,0

0,0,0,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,0,1,0,0,0,1

0,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,1,1,0,0,0

1,0,0,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,1,0,0,1

1,1,0,1,0,0,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,1,1

Once our algorithm reads the bits then it goes for the spin generation i.e the first angle for the angles generation in association with the bits. Here I made 0-360 degrees into 2 partitions 0-180 and 181-360 so 2 possibilities are either bits can be in range of 0-180 or 181-360 so 4 ways of association will be possible here.

The spin i.e first angle which decides first will be in the range of 0-360 it can be any number (angle) either 0 or 360 or any other in between these numbers (angles).

Then we get four cases emerge here:

Case1: first spin is in range of 0-180 and first bit is 0

Case2: first spin is in range of 0-180 and first bit is 1

Case3: first spin is in range of 181-360 and first bit is 0

Case4: first spin is in range of 181-360 and first bit is 1

- If the case 1 is true then we check for the 1st bit of the random generated bits then if the 1st bit is 0 then we generate random angle in the range of 0-180 else we will generate angles in the range of 181-360
- If case 2 is true then we check for the 1st bit of the random generated bits then if the 1st bit is 0 we generate random angles in the rage of 181-360 else we will generate angles in the range of 0-180
- If the case 3 is true then we check for the 1st bit of the random generated bits then if the 1st bit is 0 then we generate random angle in the range of 181-360 else we will generate angles in the range of 0-180
- If case 4 is true then we check for the 1st bit of the random generated bits then if the 1st bit is 0 we generate random angles in the rage of 0-180 else we will generate angles in the range of 181-360

Let’s have a look at the following example with the 168 bits:

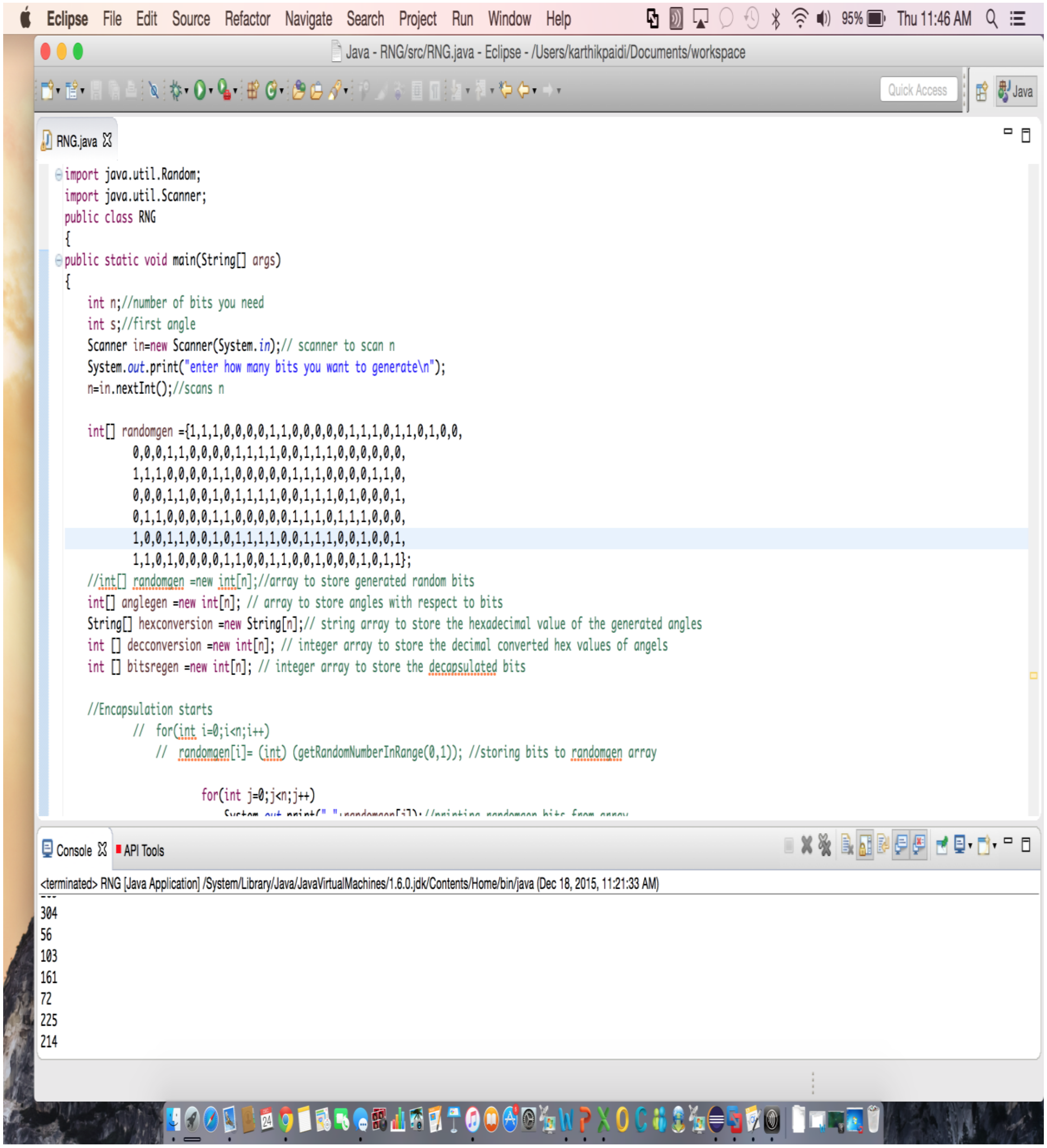


Figure 4. 168 Bits Considered for Testing

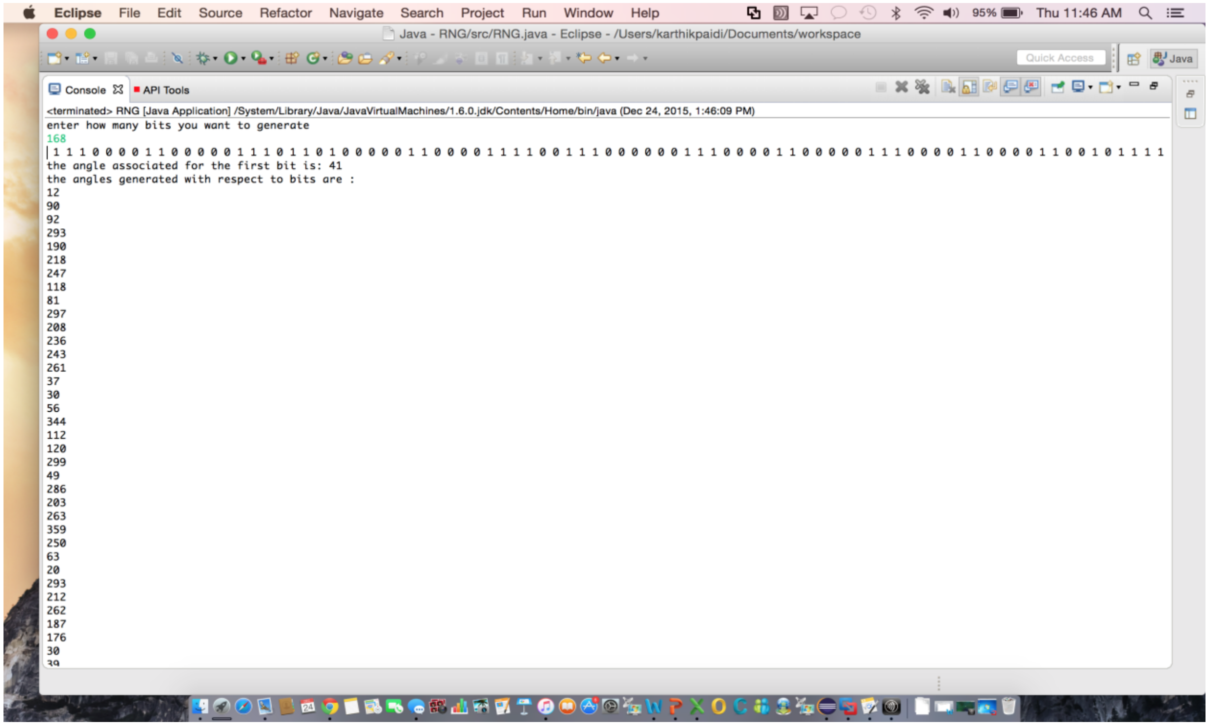


Figure 5. All Bits are Considered

If you look at the above screen shot you can see the first angle generated is 41, which is in the range of 0-180. So go back to the cases we defined above and it belongs to case 2 because the angle is in range of 0-180 and first bit is 1 then it will follow the sequence we specified for the case 2.

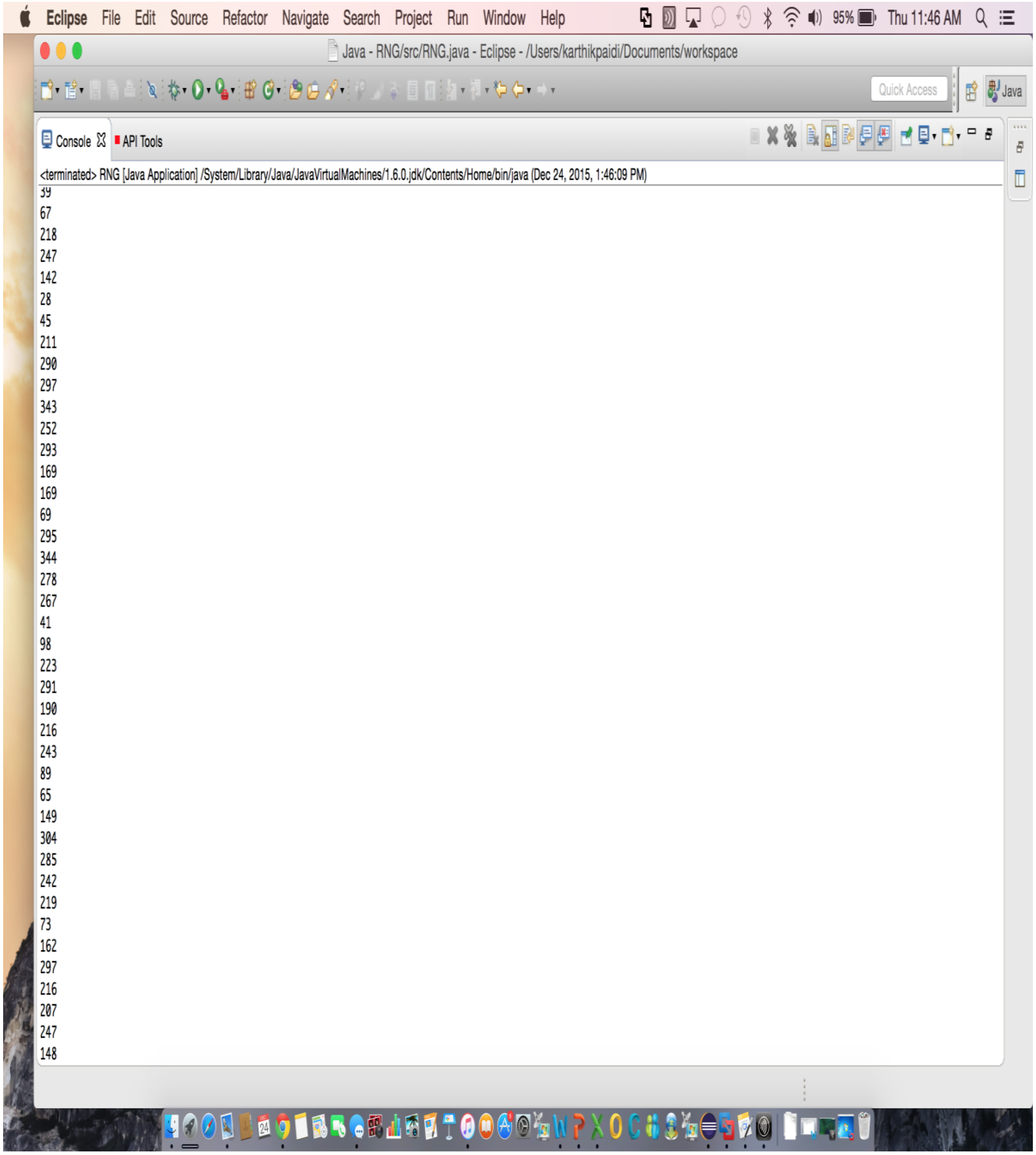


Figure 6. Shows Random Angles Generated

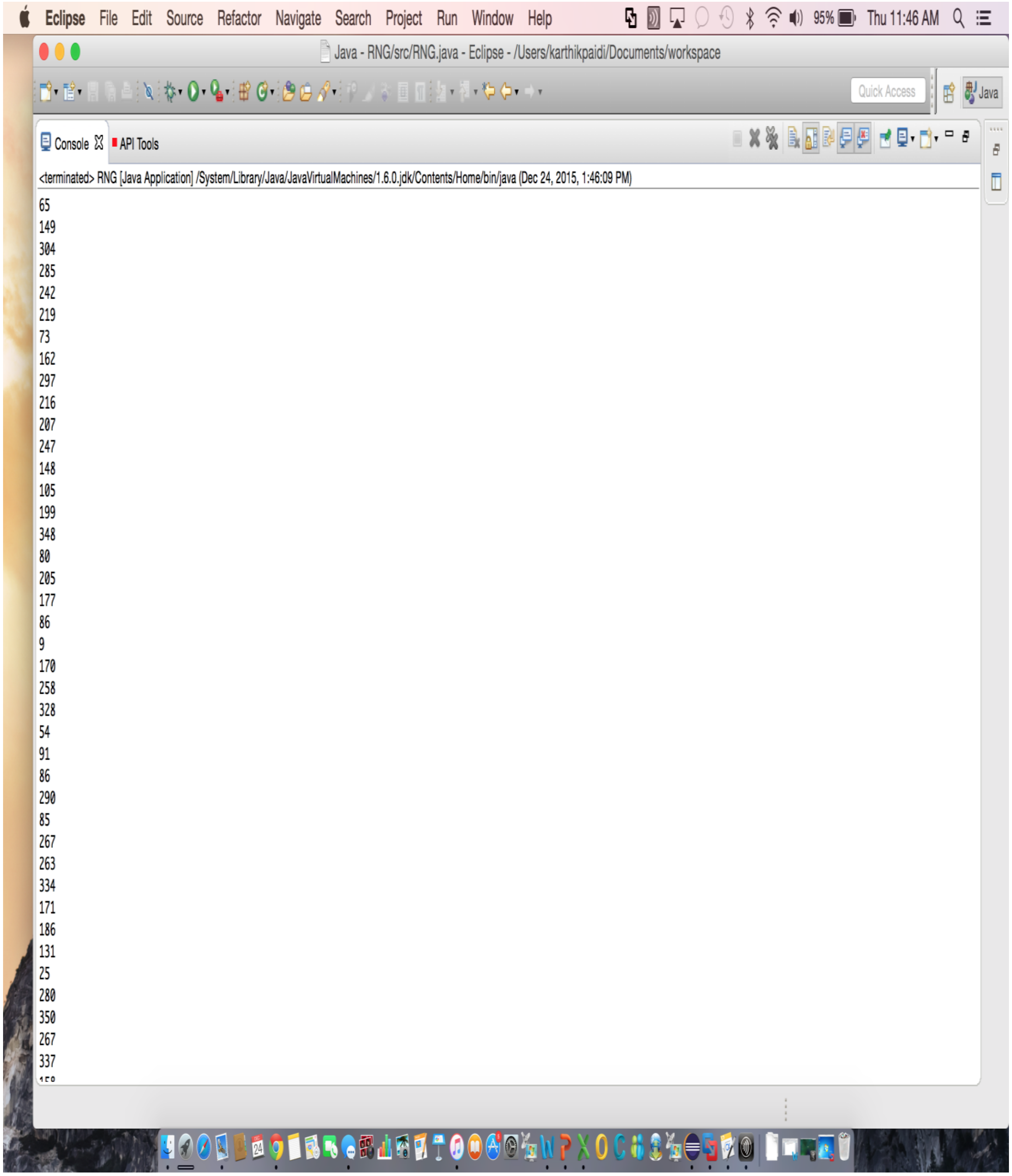


Figure 7. Shows Random Angles that are Being Generated

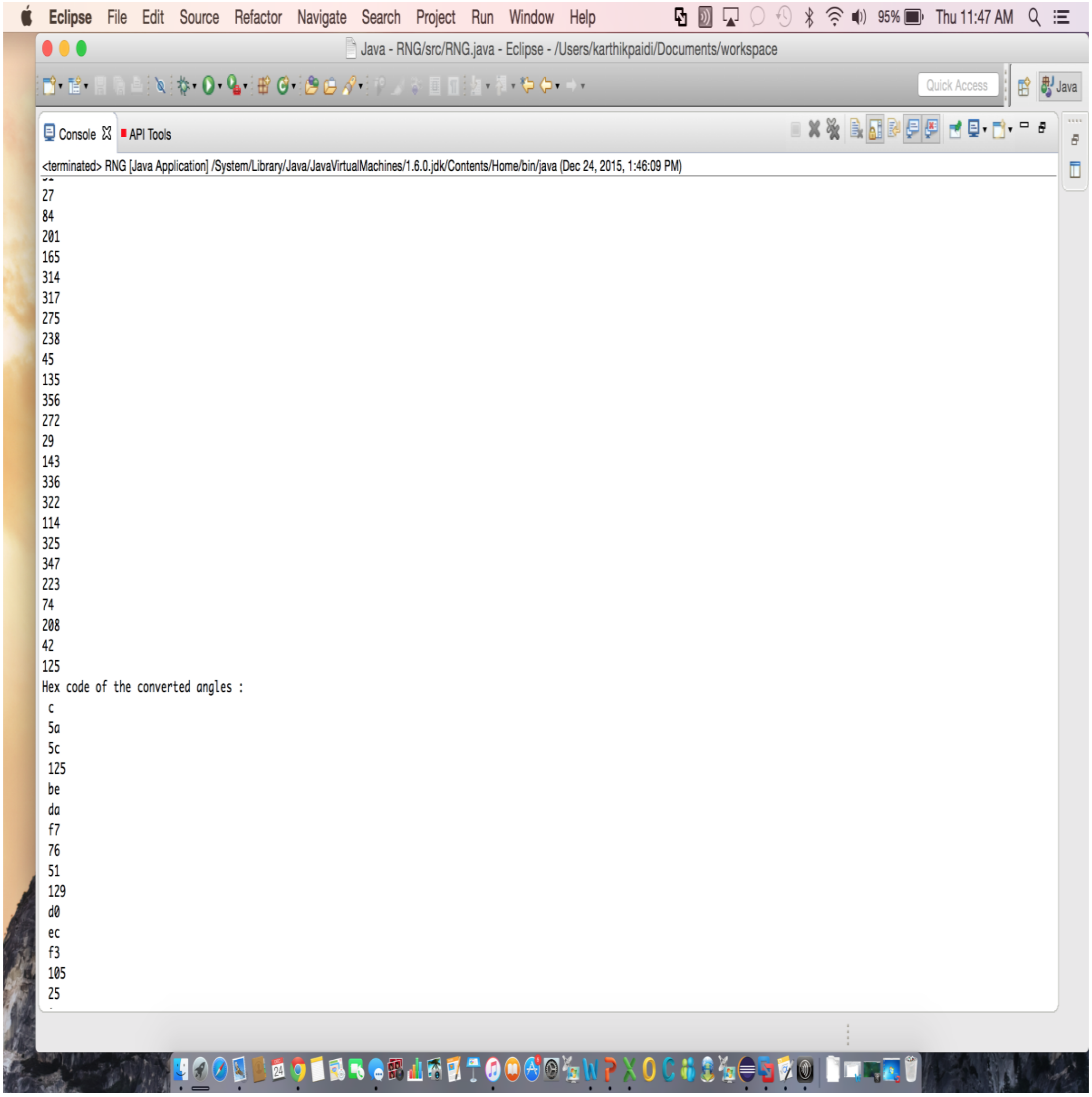


Figure 8. 168 Bits Angle Regeneration by Following Case 2

➢ Now we have to convert all those angle in to hexadecimal numbers as we want to hide or obfuscate them.

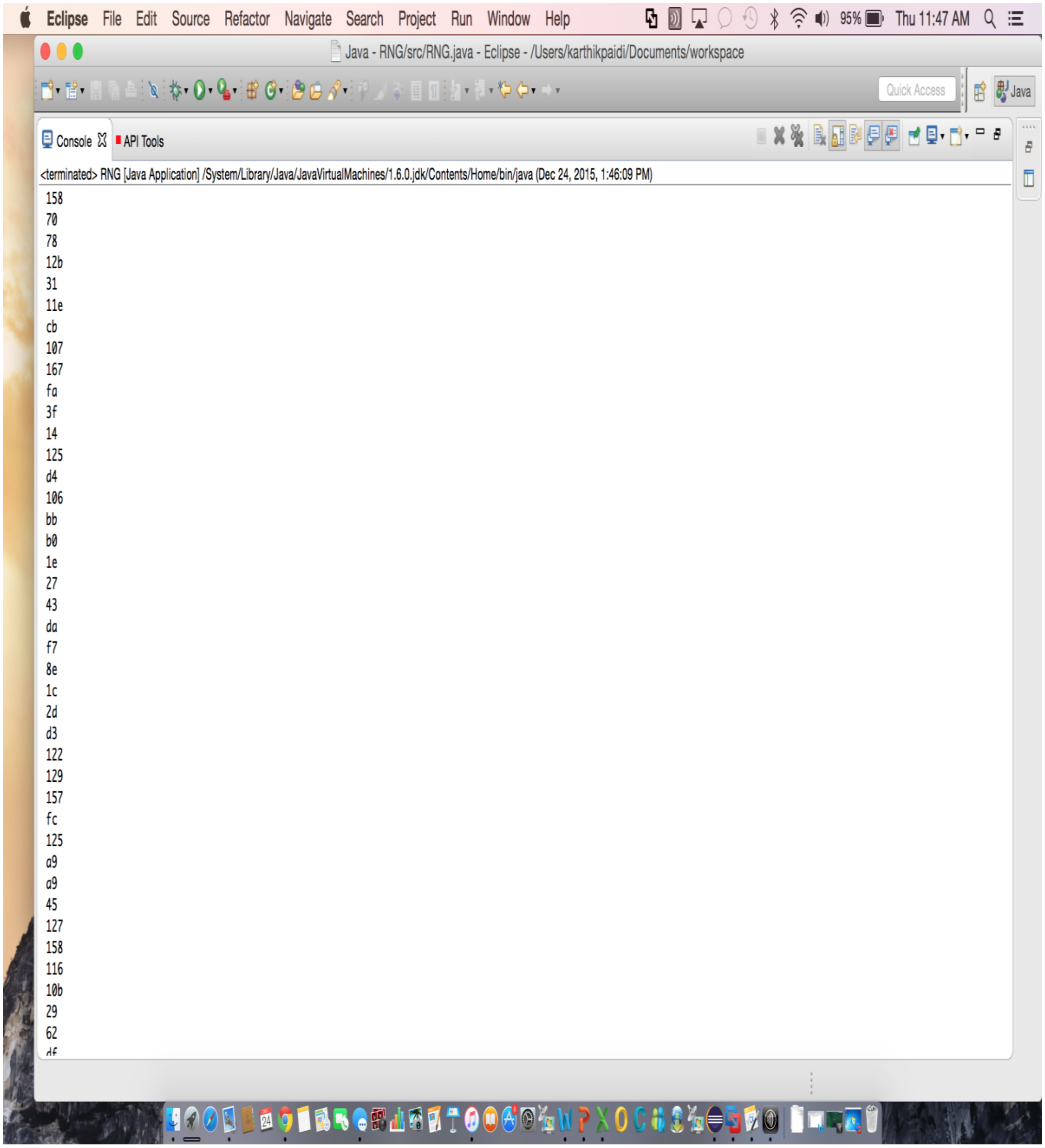


Figure 9. Conversion of the Angles to Hexadecimal Conversion

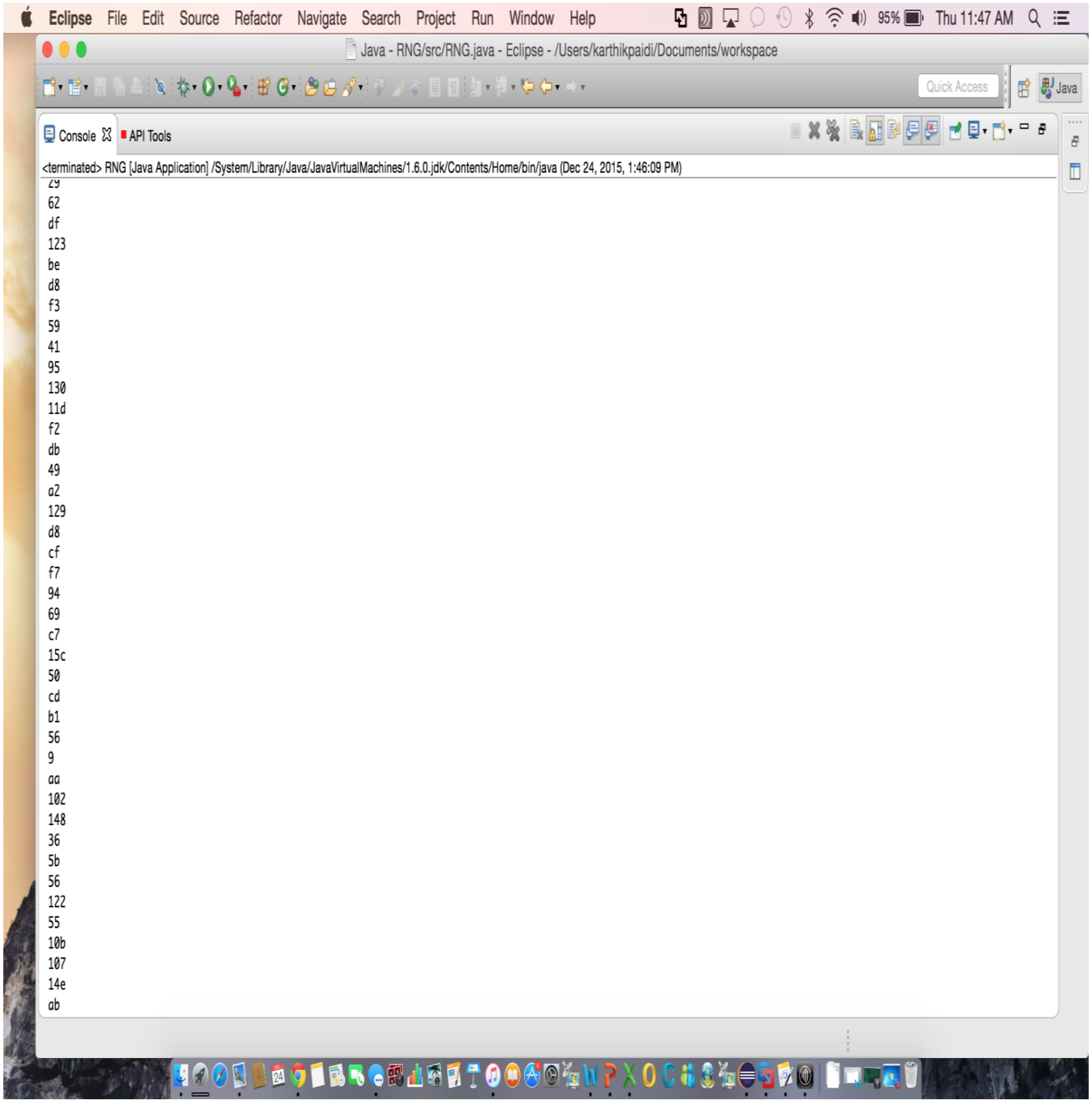


Figure 10. Angles Getting Converted into Hexadecimal

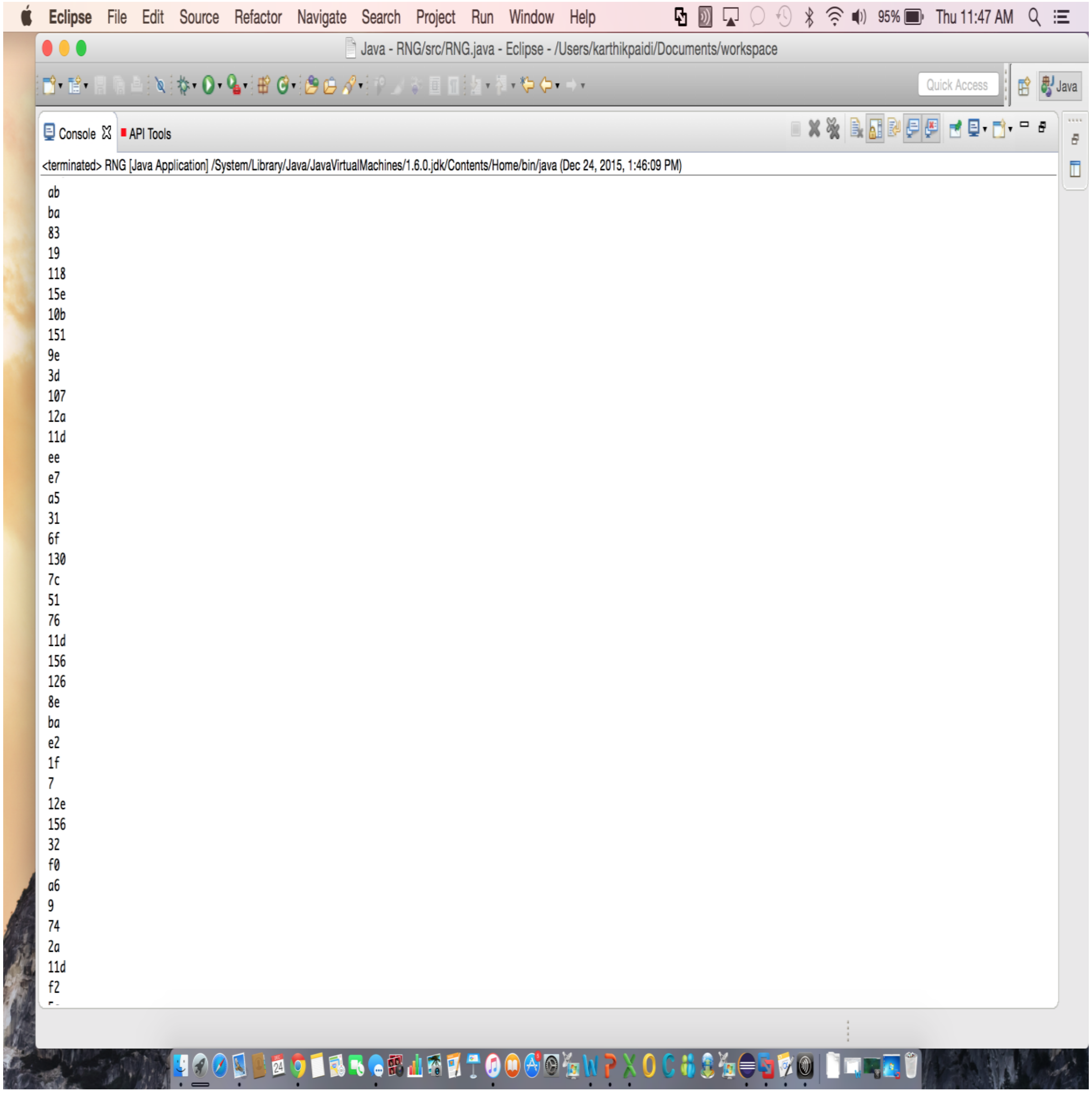


Figure 11. Additonal Angles Converted to Hexadecimal

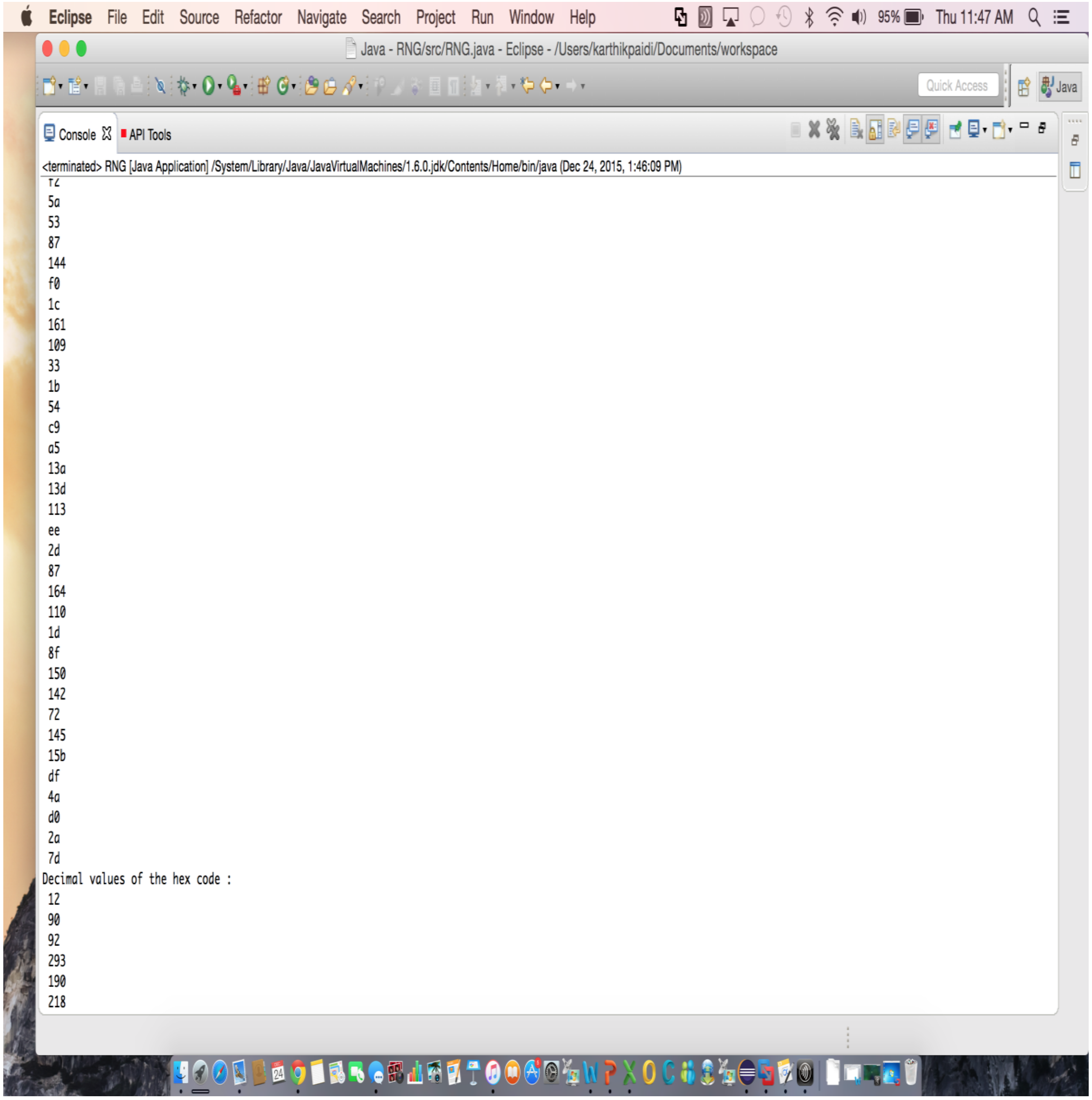


Figure 12. Final View of Angles to Hexadecimal Conversion

➢ The de-capsulation of the above requires we have to get to the original form of the bits so first we have to convert all the hexadecimal converted angles to decimal and then we have to convert all those angles in to the first generated or given random bits.

Now we will convert the hexadecimal bits in to decimal ones:

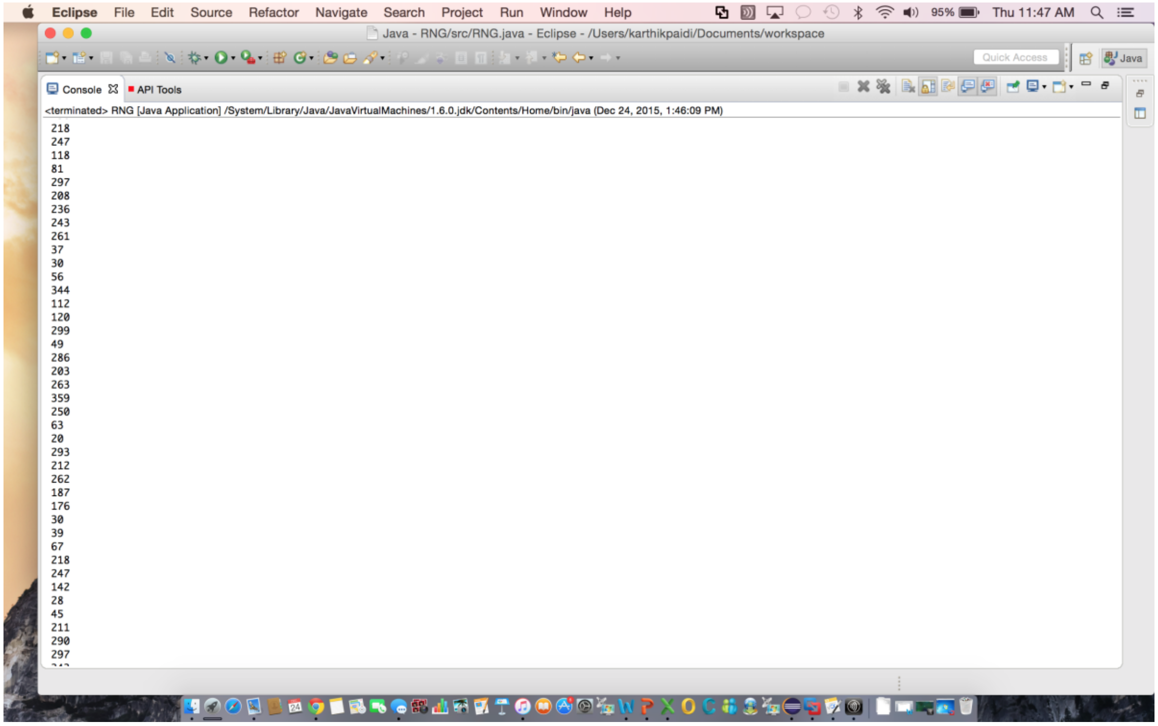


Figure 13. Shows the Conversion of the Hex to Decimal

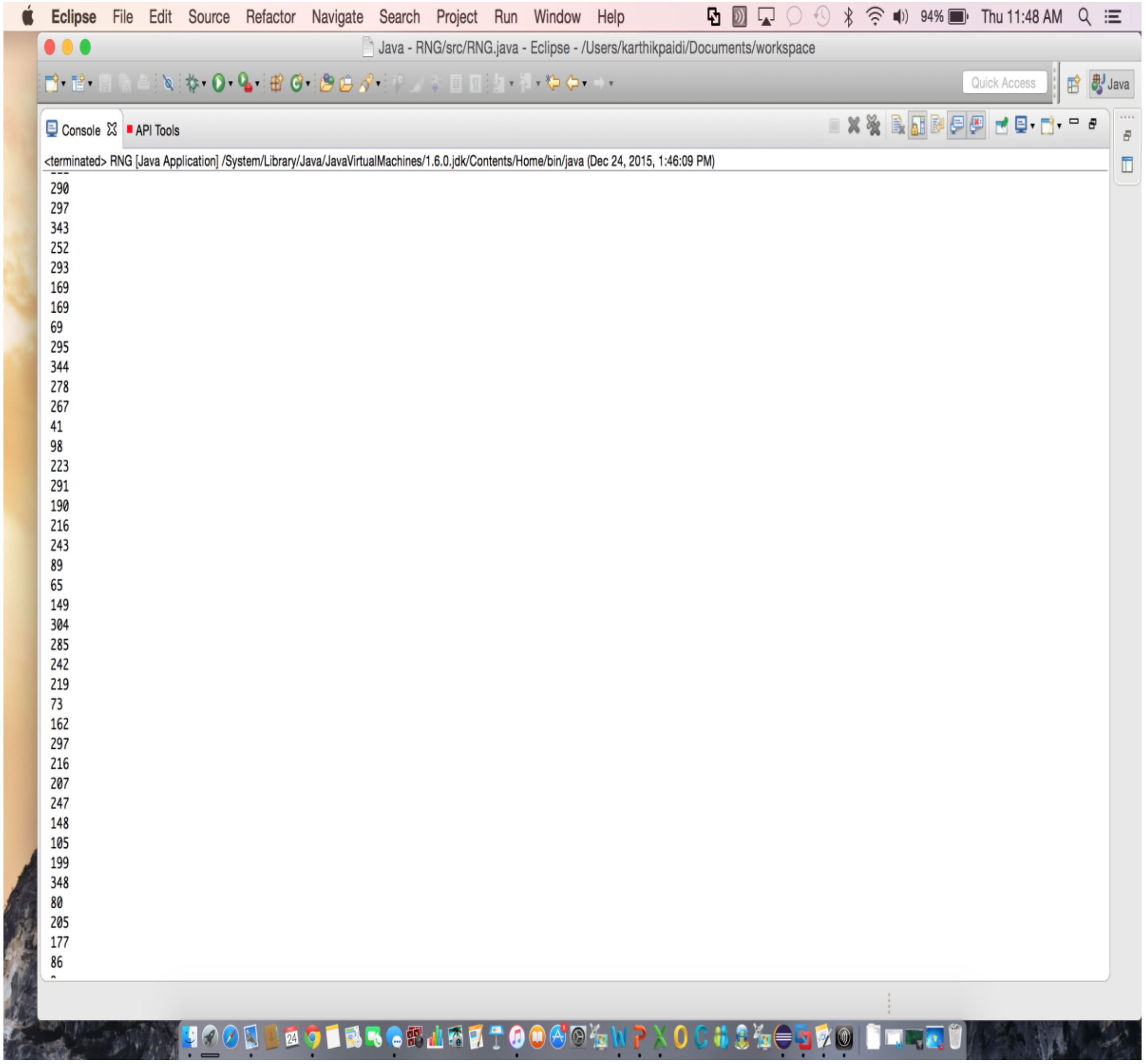


Figure 14. Continued Conversion of the Hex to Decimal

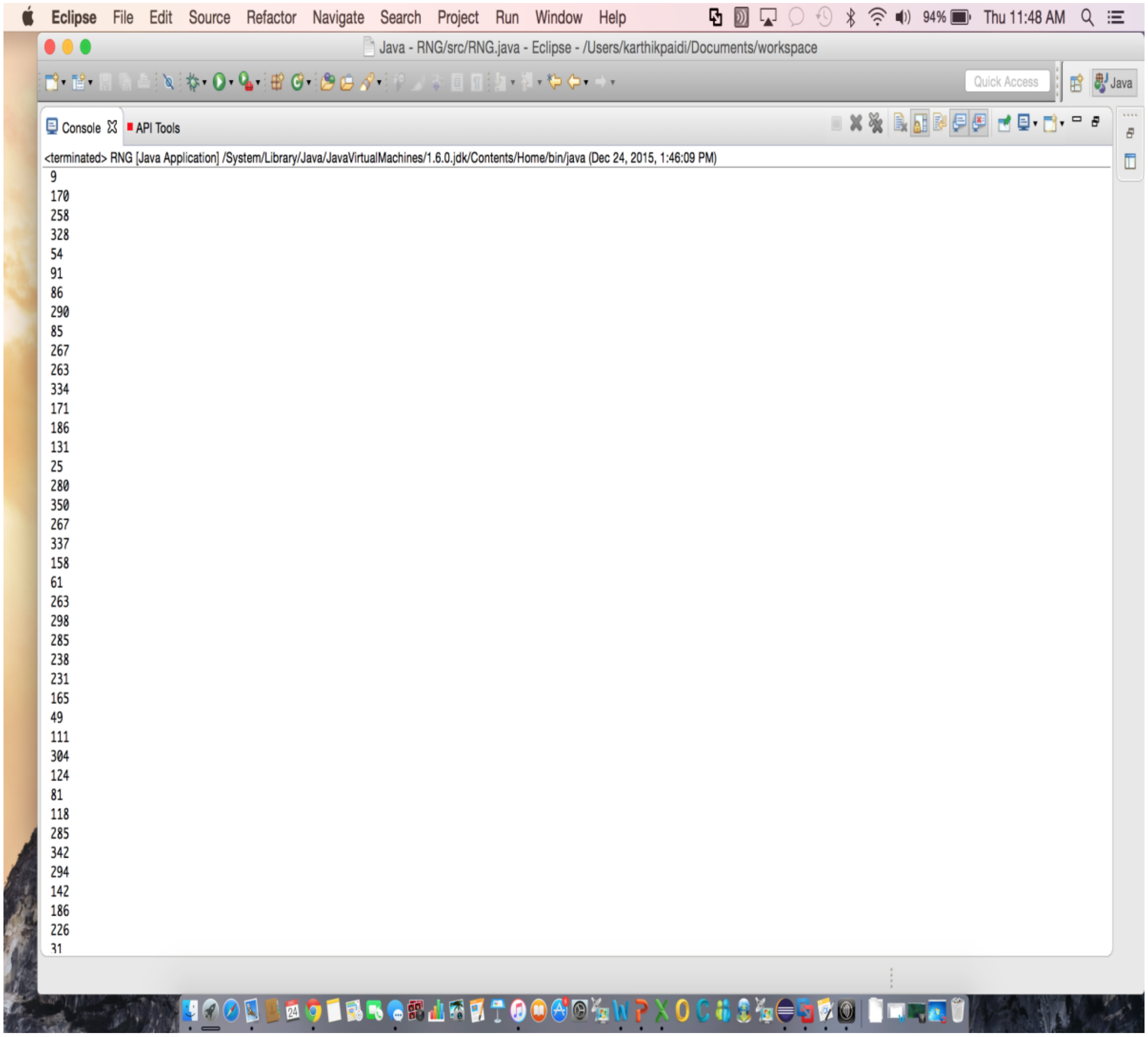


Figure 15. Conversion of the Remaining Angles to Decimal

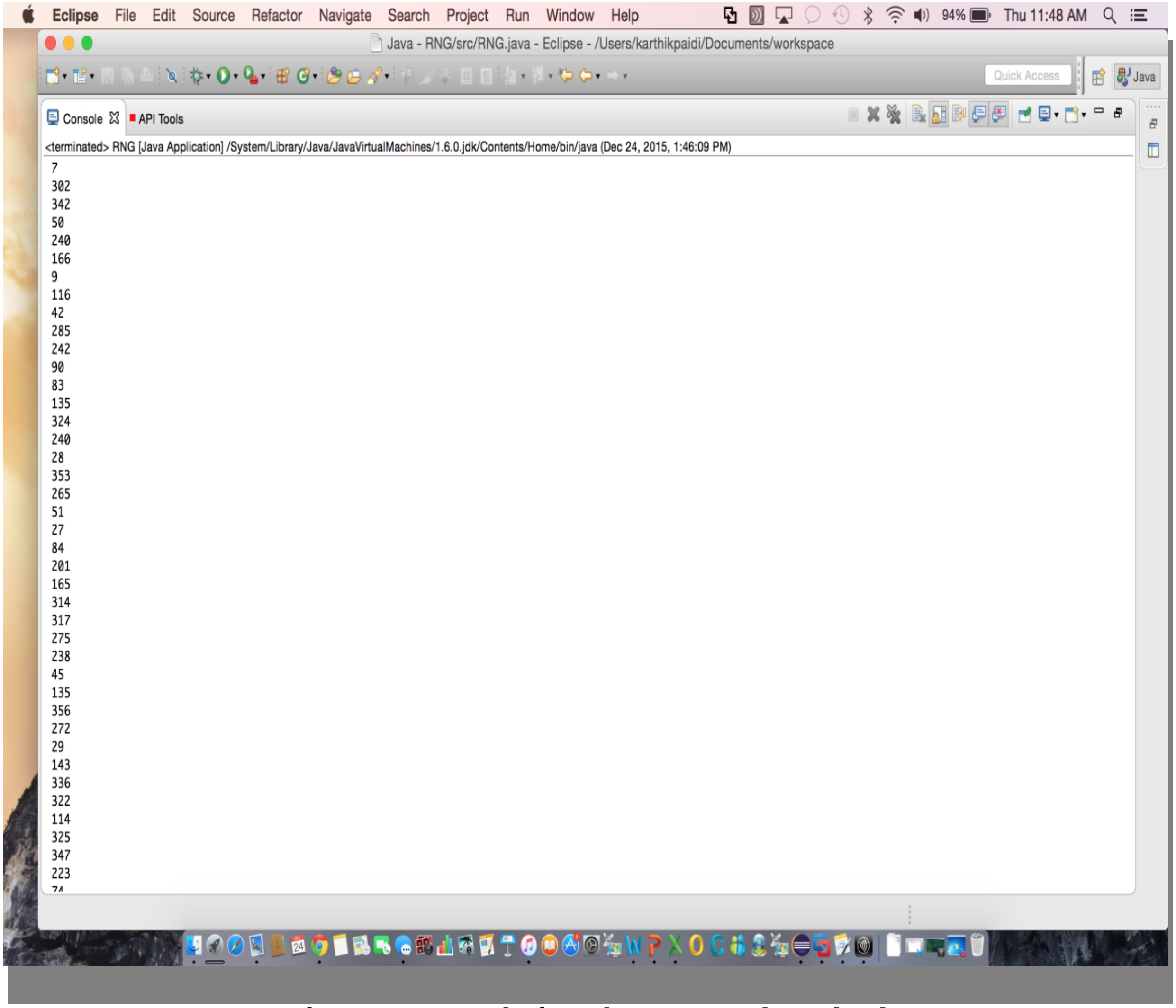


Figure 16. Conversion of Hex to Decimal of Angles Generated Randomly

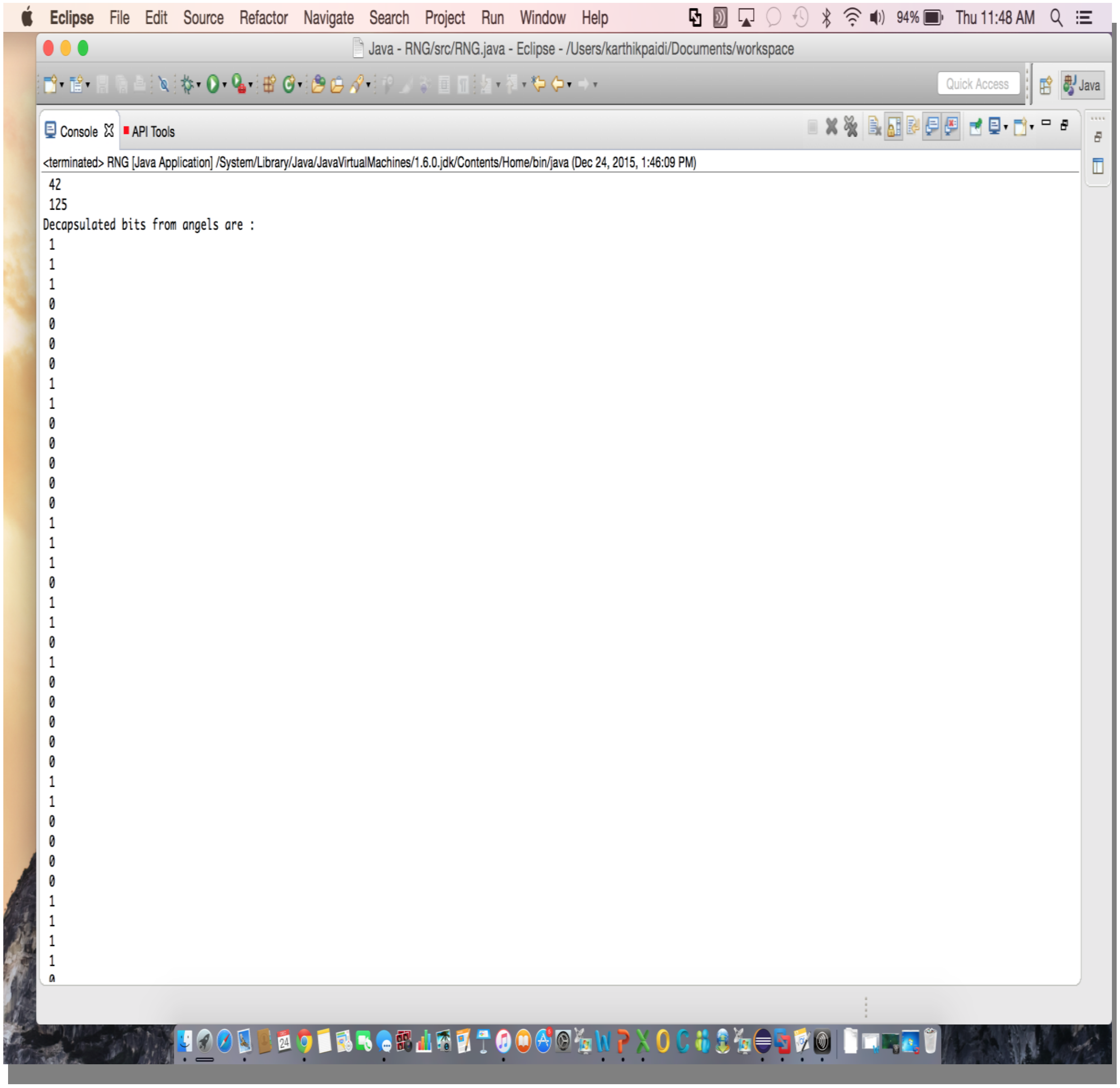


Figure 17. Decimal Values (i.e, Actual Angles of the Hex Conversion)

- Now our task is to convert those angles to normal bits for which we are given as inputs for four cases.

  The four cases are:

  Case1: first spin is in range of 0-180 and first bit is 0

  Case2: first spin is in range of 0-180 and first bit is 1

Case3: first spin is in range of 181-360 and first bit is 0

Case4: first spin is in range of 181-360 and first bit is 1

- If the case 1 is true then we check for the condition angle we got is in which range whether in 0-180 or 181-360 then if the angle is in range of 0-180 and the case 1 is true then the bit will be 0 else the bit we generate will be 1
- If the case 2 is true then we check for the condition angle we got is in which range whether in 0-180 or 181-360 then if the angle is in the range of 0-180 and the case 2 is true then the bits will be 1 else the bit we generate will be 0
- If the case 3 is true then we check for the condition angle we got is in which range whether in 0-180 or 181-360 then if the angle is in range of 181-360 and the case 3 is true then the bit will be 0 else the bit we generate will be 1
- If the case 4 is true then we check for the condition angle we got is in which range whether in 0-180 or 181-360 then if the angle is in the range of 181-360 and the case 2 is true then the bits will be 1 else the bit we generate will be 0

So from the above conditions if we look at the first spin 41 and the first bit 1 then it goes to the case 2 and after following the sequence of execution we will get the bits we first were given as input.

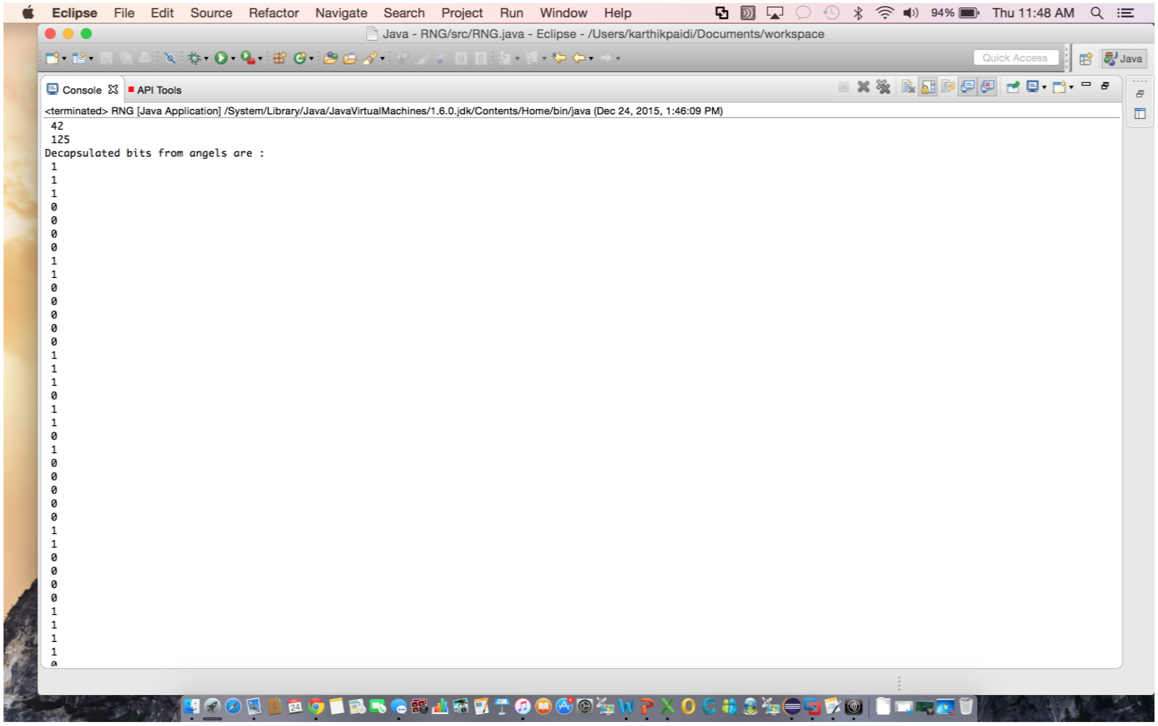


Figure 18. Conversion of Angles to Actual Bits

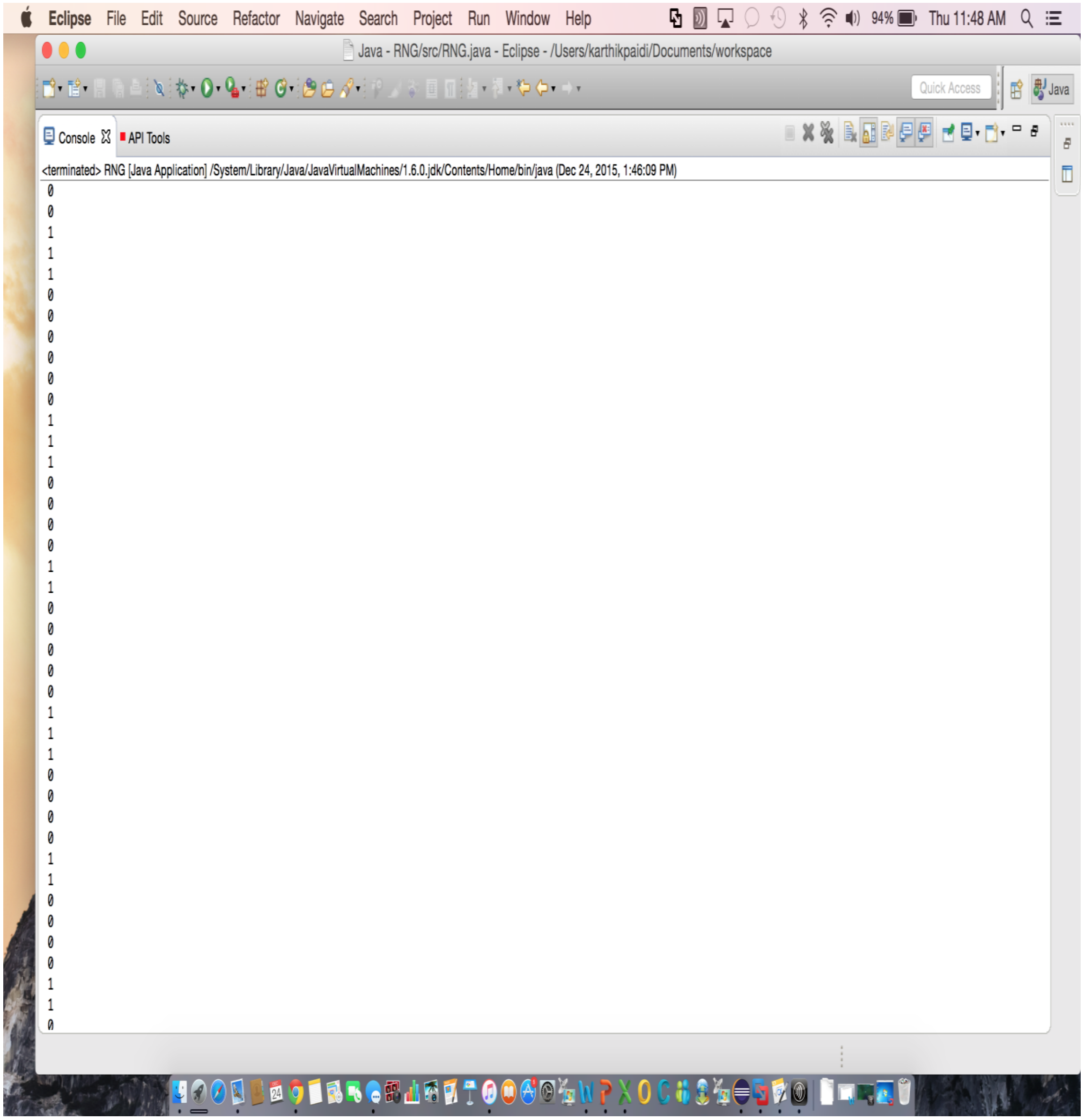


Figure 19. Conversion of the Angles to Bits

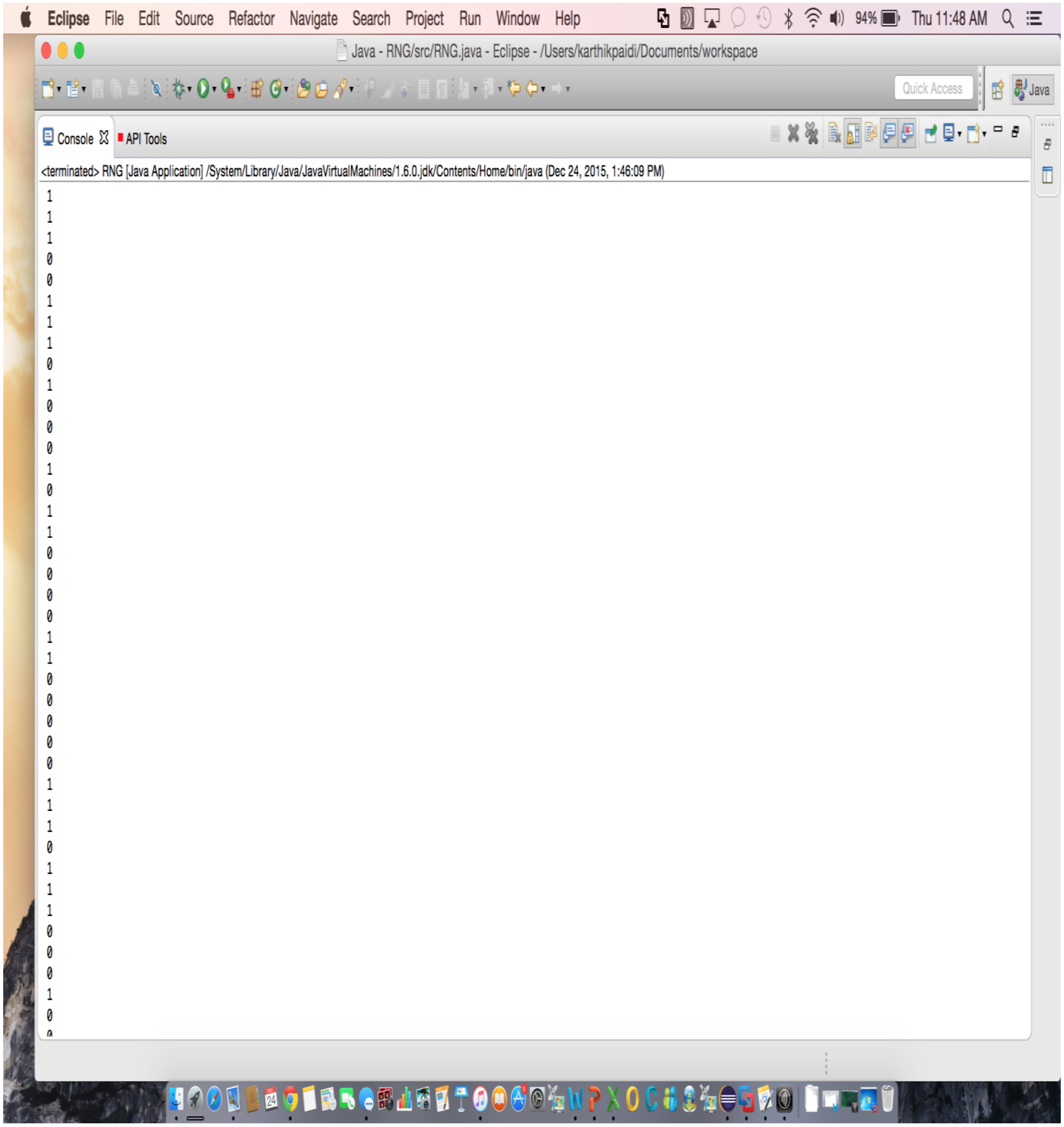


Figure 20. Conversion of Angles to Bits

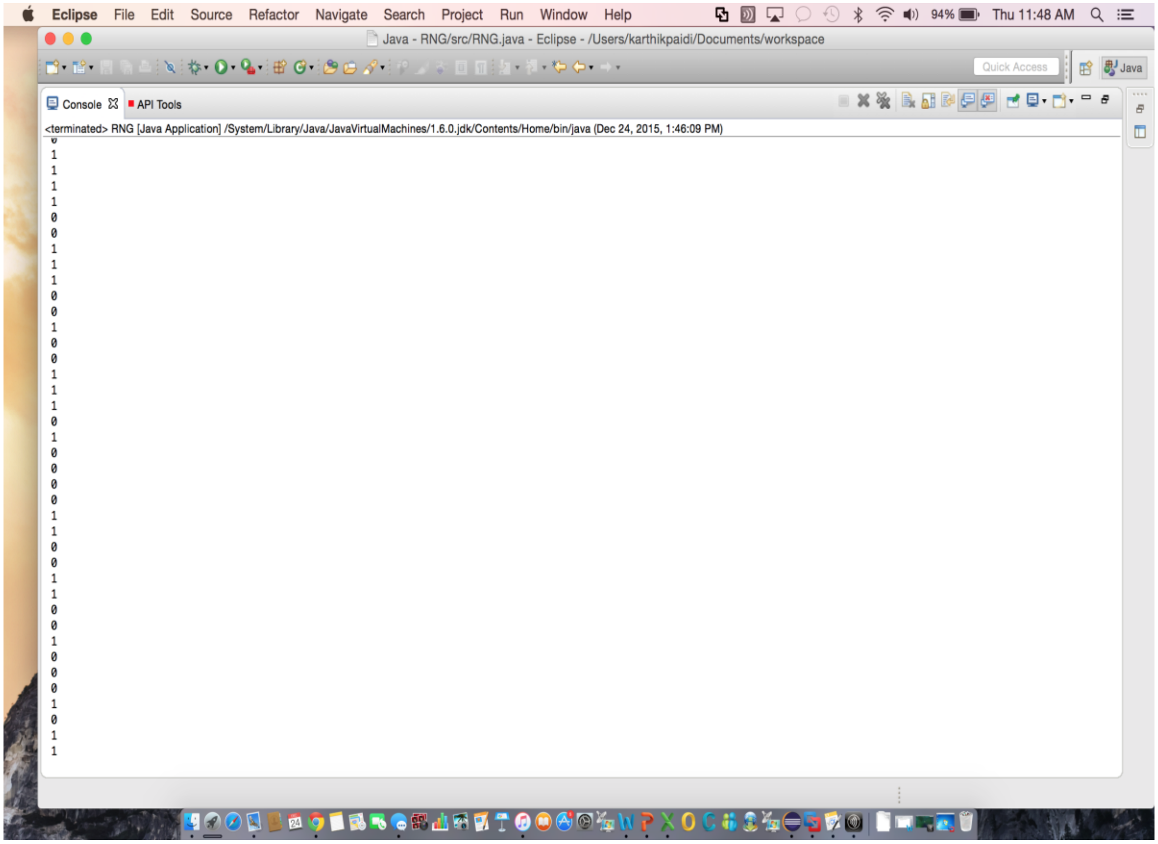


Figure 21. Actual Bits Given as the Inputs

- If we repeat the execution with the same number of or with the same bits the first spin angle will change so depending on that sequence and the case we follow will change.

Chapter 5: Conclusion and Future work

The hybrid algorithms provided in this paper is a proof of concept for many positive things emerging in the quantum security realm and can possibly stabilize current security disadvantages. The algorithms proposed here do not use any mathematical functions which might make it tough to break but can be deciphered at some point by the help of large computing machinery. The best algorithms that currently exist use the most stable format or patterns to generate keys but this algorithm is not, and the way it works and produces secure keys is also random.

Though the hybrid algorithm proposed in this paper is random regarding the generation of keys it was built using conventional implementation methodologies. The actual differences can be seen if it could be implemented practically by following the method with the help of right the physics equipment to produce truly random numbers. The future work could be applying it with proper equipment and embedding it to the current cloud infrastructure to generate keys, to generate random keys, which play a vital role in securing the conventional security algorithms.

References

[1] The Economic Implications of Moore's Law G.D. Hutcheson

[2] G.E. Moore, Lithography and the future of Moore's law. SPIE 2440, 2–17 (1995)

[3] Cramming More Components onto Integrated Circuits GORDON E. MOORE, LIFE FELLOW, IEEE

[4] http://www.dwavesys.com/

[5] http://arstechnica.com/articles/paedia/hardware/quantum.ars/1

[6] Random number generation white paper by IDQ quantis random number generation using Quantum Physics

[7] Yang Yuguang, "Some Opinion on Quantum Key Distribution Protocol [J]", *Communication Technology*, vol. 4, pp. 1021-1024, 2002

[8] Quantum computation. David Deutsch, Physics World, 1/6/92 *A comprehensive and inspiring guide to quantum computing*

[9] Two-bit heroes - Computing with quanta. The Economist Volume 338 Issue 7948 A shallow introduction to quantum computation

[10] Cue the qubits: Quantum computing- How to make the quantum computer the Economist Volume 342 Issue 8005 A very good introduction to quantum computing

[11] Liboff, R. L. *Introductory Quantum Mechanics, Fourth Edition.* San Francisco, CA: Addison Wesley, 2003

[12] Schwabl, F.; *Quantum Mechanics Second Revised Edition.* New York, NY: Springer-Verlag, 1995.

[13] Johnson,G.(2003).*A Shortcut Through Time: The path to the Quantum Computer*. New York: Alfred A Knopf

[14] C. H. Bennett and G. Brassard, *Proc. IEEE Int. Conf. Computers, Systems and Signal Processing*, Bangalore, India, December 1984, pp. 175–179.

[15] C. H. Bennett, *Phys. Rev. Lett.* 68 (1992) 3121.

[16] G. Brassard and L. Salvail "Secret key reconciliation by public discussion" Advances in Cryptology: Eurocrypt 93 Proc. pp 410-23 (1993).

[17] *Kaser, Owen; Lemire, Daniel (2013). "Strongly universal string hashing is fast". Computer Journal (Oxford University Press).*

[18] *Jordans, Frank (12 October 2007). "Swiss Call New Vote Encryption System 'Unbreakable'". technewsworld.com. Archived from the original on 2007-12-09. Retrieved 8 March 2013.*

[19] *Hughes, Richard J.; Nordholt, Jane E.; McCabe, Kevin P.; Newell, Raymond T.; Peterson, Charles G.; Somma, Rolando D. (2013). "Network-Centric Quantum Communications with Application to Critical Infrastructure Protection". arXiv:1305.0305.*

[20] Lee, C. (2015). http://arstechnica.com/science/2015/09/d-wave-unveils-new-quantum-computing-benchmark-and-its-fast/.

[21] *Kirsch, Z. (2015). Quantum Computing: The Risk to Existing Encryption Methods.* http://www.cs.tufts.edu/comp/116/archive/fall2015/zkirsch.pdf.

[22] idQuantique SA, Geneva, Switzerland, http://www.idquantique.com.

[23] MagiQ Technologies, New York, USA, http://www.magiqtech.com.

[24] Paterson, K., Piper, F. and Schack,R. (2007). Why quantum cryptography? Quantum Communication and Security, Proceedings, NATO Advanced Research Workshop, edited

by M. ˙Zukowski, S.Kilin and J. Kowalik, p. 175–180 (IOS Press, Amsterdam).Quantum cryptography network gets wireless link - info-tech - 7 June 2005 - New Scientist.

[25] Lai, H., Xue, L., Orgun, M., Xiao, J. and Pieprzyk, J. (Feb. 2015). A hybrid quantum key distribution protocol based on extended unitary operations and fountain codes. Journal of Quantum Information Processing, 14(22), pp 697-713.

[26] Nail, R. L. and Reddy, P. C. (Dec. 2015). Towards Secure Quantum Key Distribution Protocol for Wireless LANS: a Hybrid Approach. Journal of Quantum Information Processing. 14(12). Pp 4557-4574.

Appendix

1.RNG.java

```
import java.util.Random;
import java.util.Scanner;
public class RNG
{
public static void main(String[] args)
{
	int n;//number of bits you need
	int s;//first angle
	Scanner in=new Scanner(System.in);// scanner to scan n
	System.out.print("enter how many bits you want to generate\n");
	n=in.nextInt();//scans n

	int[] randomgen ={1,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,1,0,1,0,0,
				0,0,0,1,1,0,0,0,0,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,
				1,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,1,0,
				0,0,0,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,0,1,0,0,0,1,
				0,1,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,1,1,0,0,0,
				1,0,0,1,1,0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,1,0,0,1,
				1,1,0,1,0,0,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,1,1};
	//int[] randomgen =new int[n];//array to store generated random bits
	int[] anglegen =new int[n]; // array to store angles with respect to bits
	String[] hexconversion =new String[n];// string array to store the hexadecimal
value of the generated angles
	int [] decconversion =new int[n]; // integer array to store the decimal converted
hex values of angels
	int [] bitsregen =new int[n]; // integer array to store the decapsulated bits

	//Encapsulation starts
				//	for(int i=0;i<n;i++)
					//	randomgen[i]= (int)
(getRandomNumberInRange(0,1)); //storing bits to randomgen array

							for(int j=0;j<n;j++)
								System.out.print("
"+randomgen[j]);//printing randomgen bits from array
```

```
//spin angle for first bit
s=spinforfirstangle(0,360);
System.out.print("\nthe angle associated for
the first bit is:"+" "+s);
System.out.print("\n");

//decision to
make which set of angles will become 1 or 0

for(int k=0;k<n;k++)// loop to run anglegen
and randomgen arrays
{
if(s>=0
&& s<=180 && randomgen[0] == 0)// if the first spin is above 0 and below or equal to
180 and first bit 0
{

    //System.out.println("\n 0 and is in below 180 :"+s);// just to test loop is working
properly or not

    if(randomgen[k] == 0)

    {

anglegen[k]= (int) (angbetzandoneeight(0,180)); // saving the angles to array with respect
to the first spin

    }

    else

    {

    anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the
angles to array with respect to the first spin

    }
}
else if(
s>=0 && s<=180 && randomgen[0] == 1) // else if first spin is above 0 and below or
equal to 180 and first bit 1
{

    //System.out.println("\n1 and is in below 180 :"+s);// just to test loop is working
properly or not
```

```
if(randomgen[k] == 1)

{

anglegen[k]=(int) (angbetzandoneeight(0,180)); // saving the angles to array with respect to the first spin

}

else

{

anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin

}
}
else if(s>=181 && s<=360 && randomgen[0] ==0)// else if first spin is above 181 and below or equal to 360 and first bit 0
{

//System.out.println("\n0 and is in above 181 :"+s);// just to test loop is working properly or not

if(randomgen[k] == 0)

{

anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the angles to array with respect to the first spin

}

else

{

anglegen[k]= (int) (angbetzandoneeight(0,180)); // saving the angles to array with respect to the first spin

}
}
```

```
else
if(s>=181 && s<=360 && randomgen[0] == 1)// else if first spin is above 181 and below
or equal to 360 and first bit 1
{

    //System.out.println("\n1 and is in above 181 :"+s);// just to test loop is working
properly or not

    if(randomgen[k] == 1)

    {

    anglegen[k]= (int) (angbetoneeightoneandthreesixty(181,360)); // saving the
angles to array with respect to the first spin

    }

    else

    {

    anglegen[k]= (int) (angbetzandoneeight(0,180));      // saving the angles to array
with respect to the first spin

    }
}

}

//printing of the angles generated in
association with the bits are
System.out.println("the angles generated
with respect to bits are :");
for(int f=0;f<n;f++)

    System.out.println(anglegen[f]);//printing the angles

//iterative loop to convert angles to hex code
for(int a=0;a<n;a++)
{
    hexconversion[a] =
Integer.toHexString(anglegen[a]);//conversion and storage of the converted angles to
string array
}
```

```
                                        System.out.println("Hex code of the
converted angles :");
                                        //printing of the converted hexcode of the
angles
                                        for(int b=0;b<n;b++)
                                             System.out.println("
"+hexconversion[b]);//print statement to converted Hexangles


                                        //for encryption use existing algorithms
which can raise the magnitude of trustworthy security of your bit key

                                                       //decapsulation starts


                                        //revesing back from hex to decimal
values(angles genrated)

                                        for(int c=0;c<n;c++)
                                             decconversion[c] =
Integer.parseInt(hexconversion[c],16);//conversion of string to integer

                                        System.out.println("Decimal values of the
hex code :");

                                        //printing back converted decimal
                                        for(int d=0;d<n;d++)
                                             System.out.println("
"+decconversion[d]);//printing of the decimal values of the hex

                              //converting back to bits generated
                                        for(int g=0;g<n;g++)// loop to run anglegen
and randomgen arrays
                                        {
                                                                 if(s>=0
&& s<=180 && randomgen[0] == 0)//spin in 0-180 and first bit is 0
                                                                 {


     if(decconversion[g]>= 0 && decconversion[g] <=180)// angle generated is in
range of 0-180

     {

bitsregen[g]= 0 ;// depending on first bit 0
```

```
}
else
{
bitsregen[g]= 1;// if first bit its not zero they fall under 1
}
}
else if(
s>=0 && s<=180 && randomgen[0] == 1) // else if spin in 0-180 and first bit is 1
{

if(decconversion[g]>= 0 && decconversion[g] <=180)// angle generated is in
range of 0-180
{
bitsregen[g]= 1 ;// depending on first bit 1
}
else
{
bitsregen[g]= 0;// if first bit its not one they fall under 0
}
}
else
if(s>=181 && s<=360 && randomgen[0] ==0)// else if spin in 181-360 and first bit is 0
{

if(decconversion[g]>= 181 && decconversion[g] <=360)// angle generated is in
range of 0-180
{
bitsregen[g]= 0 ;// depending on first bit 0
```

```
}

else

{

bitsregen[g]= 1;// if first bit its not zero they fall under 1

}
}
else
if(s>=181 && s<=360 && randomgen[0] == 1)// else if spin in 181-360 and first bit is 0
{


if(decconversion[g]>= 181 && decconversion[g] <=360) // angle generated is in
range of 0-180

{

bitsregen[g]= 1 ;// depending on first bit 1

}

else

{

bitsregen[g]= 0;// if first bit its not one they fall under 0

}
}

}

System.out.println("Decapsulated bits from angels are :");
//printing the decapsulated random gen bits
for(int e=0;e<n;e++)
System.out.println(" " +
bitsregen[e]);//prints the exact bits used for key generated
}// main class close

// function to generate random bits
private static int getRandomNumberInRange(int min, int max) {
```

```
		//exception case for number generation logic
		if (min >= max) {
			throw new IllegalArgumentException("max must be greater than
min");
		}

		Random r = new Random();//random generating internal function

		return r.nextInt((max - min) + 1) + min;// returning of generated numbers
	}

		// function to generate random number i.e first angle
			private static int spinforfirstangle(int min, int max) {

				//exception case for number generation logic
				if (min >= max) {
					throw new IllegalArgumentException("max must
be greater than min");
				}

				Random r = new Random();//random generating internal
function

				return r.nextInt((max - min) + 1) + min;// returning of
generated first associated angle
			}

				//function to generate angles between 0 and 180
				private static int angbetzandoneeight(int min, int max) {

						//exception case for number generation logic
						if (min >= max) {
								throw new
IllegalArgumentException("max must be greater than min");
							}

							Random r = new Random();//random
generating internal function

							return r.nextInt((max - min) + 1) +
min;// returning of generated first associated angle
					}
```

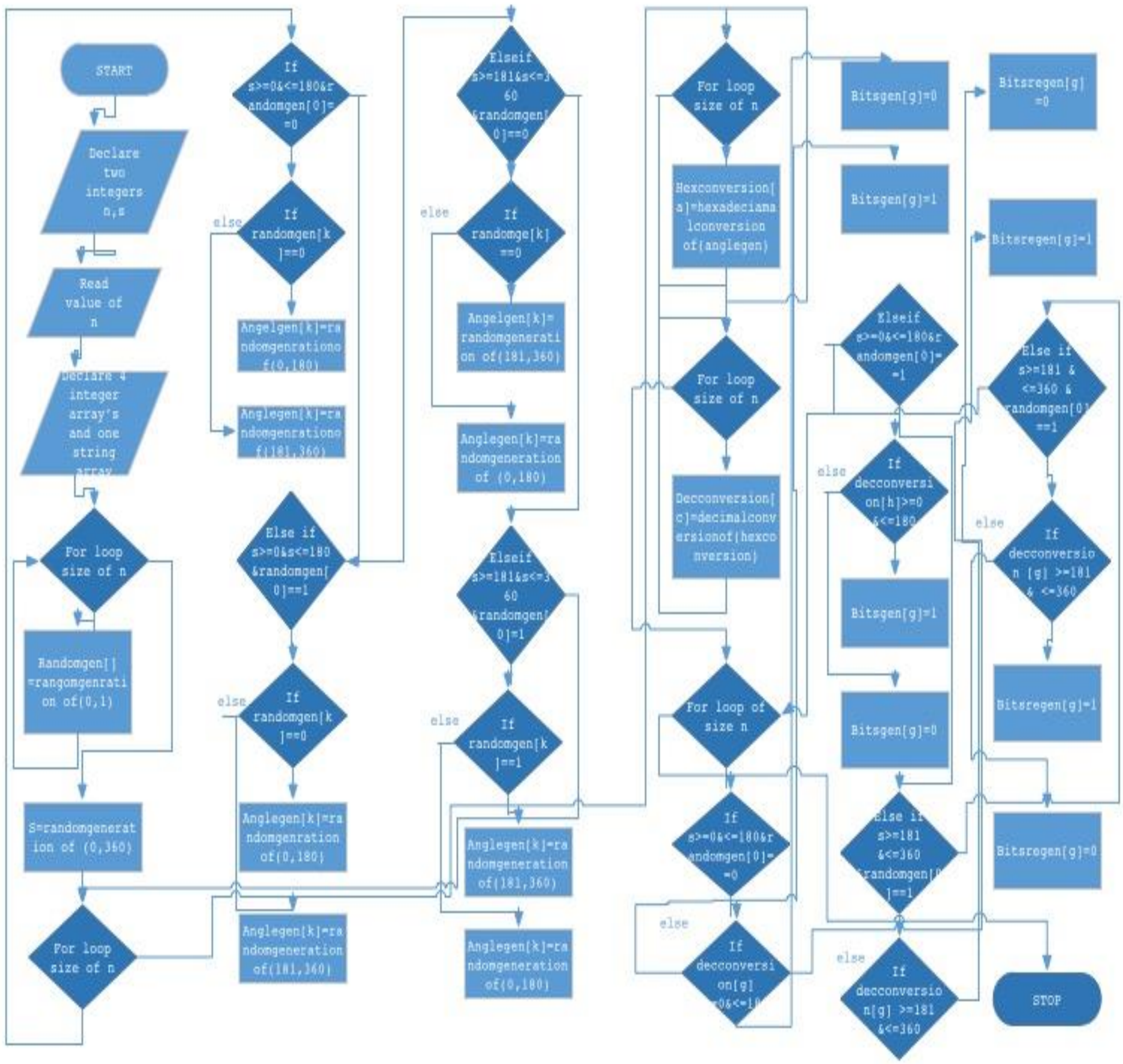
```
//function to generate angle between 181 and
360
private static int
angbetoneeightoneandthreesixty(int min, int max) {

//exception case for
number generation logic
if (min >= max) {
throw
new IllegalArgumentException("max must be greater than min");
}


Random r = new Random();//random generating internal function


return r.nextInt((max - min) + 1) + min;// returning of generated first associated
angle
}
}//class close
```

2.Flowchart



3.Randmon_generator_Algorithm

Step1: START

Step2: Declare two integer type variables: n, s

Step3: Declare 4 integer arrays, i.e. randomgen, anglegen, decconversion, bitsregen and one string type array hexconversion

Step4: Using the scanner method scan the number of bits the user wants to generate i.e. n

Step5: Write a For loop that runs for n

For loop starts

Step6: To generate the random bits call a method that can generate random bits of a given size. (here it is a recursive function)

Step7: Define a method to get random binary numbers within a range with min, max as integer arguments and inside use Random () predefined and return the bits generated in the range of 0,1 (here min =0 and max =1)

Step8: Store the randomly generated bits to an array randomgen

For loop closes when the bits are generated for the given size

Step9: Run a spin which selects any one number randomly from 0 to 360 and store the spin in to (S) integer and to this random number define a method spinforfirstangle with min=0, max=360 as arguments

Step10: To generate respected angles for the generated random bits declare for loop for size n.

For loop starts

Step11: Check for the condition if spin s>=0 and s<=180 and generated first random bit randomgen [0] == 0 then check for another condition if randomgen of kth bit == 0 then store the angles into array anglegen and those angles are from 0-180 to generate random angles form 0-180 write the same kind of function in Step7 but the min and max are 0,180 i.e all the 0's of the generated bits are now in range of 0-180

Step12: Else case is for the 1's where those bits belong to 181-360 to generated the random angles from 181-360 use method angbetoneeightoneandthreesixyt and the arguments are 181 and 360, i.e. all the 1's belongs to 181-360 in this case

Step13: Else if check for the condition spin generated is in s>=0 and s<=180 and first randomly generated bit randomgen [0] == 1 then check for condition if randomgen [kth bit] == 1 then store the angles generated from 0-180 and stored into anglegen and generate those angles using method in Step11 i.e. all the 1's generated will be in the range of 0-180

Step14: Else case is for the 0's where those bits belongs to 181-360 to generated the random angles from 181-360 use method angbetoneeightoneandthreesixyt and the arguments are 181 and 360, i.e. all the 0's belongs to 181-360 in this case

Step15: Else if check for the condition if spin s>=181 and s<=360 and generated first random bit randomgen [0] == 0 then check for another condition if randomgen of kth bit == 0 then store the angles into array anglegen and those angles are from 181-360 to generate random angles form 181-360 write the same kind of function in Step7 but the min and max are 181, 360 i.e all the 0's of the generated bits are now in range of 181-360

Step16: Else case is for the 1's where those bits belongs to 0-180 to generate the random angles from 0-180 use method anglebetzandoneeight and the arguments are 0 and 180, i.e. all the 1's belongs to 0-180 in this case

For loop is closed

Step17: Run a for loop of size n to convert the generated angles stored in anglegen array to hexadecimal

Step18: Write the predefined or new user defined function to convert the decimal code of angles to hexadecimal code and store them to hexconverion array

Step19: Now we have to reverse the procedure to get the old bits for which, we converted the hexadecimal code to decimal and store them in to an integer array

Step20: After getting the decimal values declare a for loop of size n to convert those angles back to normal randomly generated bits

For loop starts

Step21: Check for the condition if the first spin s>=0 and s<=180 and first bit randomgen [0] == 0 then check for condition if decconversio [gth bit] >=0 and<=180 then all the angles in the range of 0-180 are 0's else the bits that are going to be regenerated will be 1's for all the other angles i.e (181-360)

Step22: Else if check for the condition if the first spin is s>=0 and<=180 and first bit randomgen [0] ==1 then check for the condition if decconversion [gth bit] >=0 and <=180 then all the angles in range of 0-180 will become as 1's else the rest of the angles will become 0's i.e (181-360)

Step23: Else if the first spin generated s>=181 and <=360 and the first bit generated randomgen [0] ==0 then check for the condition if decconversion [gth bit] >=181 and <=360 then all the angles in the range of 181-360 converted to 0's else other angles are converted to 1's i.e 0-180

Step24: Else if the first spin generated s>=181 and <=360 and the first bit generated randomgen [0] ==1 then check for the condition if decconversion [gth bit] >=181 and <=360 then all the angles in the range of 181-360 converted to 1's else other angles are converted to 0's i.e 0-180

For loop closed